CENTRE
DE RECHERCHE
SUR LES TRANSPORTS
(CRT)
CENTRE
FOR RESEARCH
ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

MÉMOIRE D'ÉTUDIANT

C.P. 6128, SUCCURSALE A
MONTRÉAL (QUÉBEC)
CANADA
H3C 3J7

TÉLÉPHONE : (514) 343-7575
TÉLEX : 05-24146 BIBPOLYTEC
TÉLÉCOPIEUR (FAX) : (514) 343-7121

CRT-677

MISE AU POINT D'UN SYSTÈME INTERACTIF-GRAPHIQUE POUR LA PLANIFICATION TACTIQUE DU TRANSPORT DES MARCHANDISES

par

Jean-François Mondou

Mémoire de maîtrise, département d'Informatique et de Recherche opérationnelle, Université de Montréal, codirigé par Teodor Gabriel Crainic et Sang Nguyen.

Centre de recherche sur les transports - Publication #677

Décembre 1989

90-03006/

Dépôt légal - 1er trimestre 1990

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Mise au point d'un système interactif-graphique pour la planification tactique du transport des marchandises

présenté par:

Jean-François Mondou

a été évalué par un jury composé des personnes suivantes:

_____	:	président-rapporteur
_____	:	directeur de recherche
_____	:	co-directeur
_____	:	membre du jury

Mémoire accepté le: _____

SOMMAIRE

L'objectif général de ce programme de recherche porte sur l'étude et la mise au point d'un prototype d'un système interactif graphique, intégrant un modèle et des algorithmes de recherche opérationnelle, pour la planification tactique du transport interurbain des marchandises.

Le modèle développé par notre équipe représente une approche de résolution globale pour un problème de planification tactique et a déjà été utilisé avec succès pour résoudre deux problèmes complexes : un problème de transport par chemin de fer et un problème de camionnage en lots brisés. Cette méthodologie a été expérimentée à l'aide de données provenant d'importantes entreprises de transport tant canadiennes qu'internationales.

Notons que le problème que nous étudions possède un nombre élevé de solutions possibles avec sensiblement le même coût total. Dans ce contexte, il est intéressant pour le planificateur de disposer d'un système interactif graphique. Il peut interagir avec le système pour choisir, parmi plusieurs solutions, celle qui lui semble la plus favorable pour ensuite la manipuler afin qu'elle respecte les règles et critères non inclus dans le modèle. Il peut également visualiser et analyser l'impact sur l'opération et le rendement du système (congestion, activités aux noeuds, qualité du service, etc.) de diverses décisions stratégiques (ouverture, fermeture ou modification d'installations, modifications de la demande, fusion d'entreprises, etc.).

À notre connaissance, il n'existe aucun système interactif graphique adapté aux besoins d'un instrument de planification tactique des réseaux de transport interurbain des marchandises. On en trouve cependant plusieurs pour la planification des réseaux de transport urbain de personnes. Bien qu'il soit généralement possible d'utiliser ces derniers pour l'analyse des

mouvements de marchandises au niveau stratégique national, ils se prêtent difficilement aux besoins tactiques de planification du transporteur interurbain de marchandises.

Le présent mémoire a précisément pour objectif de combler, au mieux, cette lacune.

Présentement, l'étude du noyau méthodologique est presque complétée, et nous travaillons depuis presque deux ans à la représentation et à l'affichage des éléments du problème (réseau, services, itinéraires, etc.). L'outil, que nous appelons TACTIC, se compose d'un noyau méthodologique, d'une interface interactive-graphique ainsi que d'une banque de données.

L'objectif général de ce mémoire s'inscrit dans le cadre du projet TACTIC et il vise à intégrer les deux aspects de la programmation mathématique et informatique du problème. Nous ne prétendons pas, ici, à produire un logiciel complet, c'est là une tâche qui dépasse les objectifs d'un mémoire de maîtrise. Nous visons plutôt à établir les conditions pour qu'un tel logiciel soit opérationnel et à créer les fondations pour les travaux futurs.

Nous avons donc commencé la mise au point du prototype en concentrant d'abord nos efforts à l'analyse de la structure du modèle et des algorithmes de recherche opérationnelle. Plus précisément, nous avons abordé les algorithmes du chemin le plus court. Nous avons étudié les performances des nouveaux algorithmes par rapport à celui déjà utilisé pour l'optimisation de l'acheminement des marchandises à travers le réseau de services. Nous avons aussi examiné les algorithmes de réoptimisation pour la détermination des chemins les plus courts entre tous les noeuds du réseau.

À notre connaissance, il n'existe aucun résultat d'analyse numérique pour l'algorithme SRHEAP dû à Ahuja et al. (1988). Dans ce mémoire, nous pallions à cette lacune en rapportant plusieurs résultats expérimentaux.

Nous donnons le détail des implantations en langage C de huit algorithmes du chemin le plus court. Aussi, nous discutons des impacts de la programmation en langage C sur l'efficacité de ces algorithmes.

Parallèlement, une partie importante de ce projet de recherche fut consacrée à l'étude des problèmes particuliers de représentation et d'affichage des éléments intervenant lors du processus de planification des opérations de transport des marchandises. Nous avons implanté une bibliothèque de fonctions graphiques en langage C et nous discutons des solutions que nous apportons aux difficultés de représentation et d'affichage des éléments du modèle. Nous avons pris soin d'apporter des solutions générales à ces problèmes de manière à ce qu'elles puissent être facilement adaptées à d'autres problèmes connexes. Notre discussion est illustrée à l'aide de plusieurs sorties graphiques provenant d'une application actuelle au transport des marchandises par chemin de fer.

Aussi, nous voyons à implanter une structure de données favorable à l'exécution des algorithmes ainsi qu'à l'accès rapide des informations pour les manipulations graphiques. Enfin, nous développons les premières interfaces de communication avec l'usager tels le langage des commandes et la structure des dialogues.

Mots clés : Planification tactique du transport des marchandises, système interactif-graphique, langage C, chemins les plus courts, implantations avec des tableaux et des pointeurs.

TABLE DES MATIÈRES

	Page
SOMMAIRE	iii
TABLE DES MATIÈRES	vi
LISTE DES TABLEAUX	ix
LISTE DES FIGURES	x
INTRODUCTION	1
CHAPITRE 1 : Situation du problème	6
1.1 - Présentation du problème	6
1.2 - Éléments du problème	8
1.3 - L'approche interactive-graphique	12
1.4 - Principes de développement	14
CHAPITRE 2 : Éléments de programmation mathématique	17
2.1 - Modélisation	17
2.2 - Résolution	20
2.3 - Importance des chemins les plus courts	22
CHAPITRE 3 : Chemin le plus court : Revue de la bibliographie	26
3.1 - Notations-définitions	27
3.2 - Problème du chemin le plus court	30
3.3 - Condition d'existence d'un chemin le plus court	31
3.4 - Principe d'optimalité d'un chemin le plus court	32
3.5 - Chemins optimaux en programmation linéaire	35
3.6 - Méthode du simplexe	39
3.7 - Approche unificatrice	45
3.8 - Recherche du minimum d'abord	50
3.8.1 - Algorithme SDKSTR	54
3.8.2 - Arbres partiellement ordonnés, algorithme SHEAP	54
3.8.3 - Adressage dispersé - Algorithme SDIAL	57
3.8.4 - Algorithme SRHEAP	60
3.8.5 - Transformation des longueurs	65
3.9 - Recherche en largeur d'abord - Algorithme LQUEUE	66

Page

3.10 - Recherche en profondeur d'abord 69
3.11 - Listes à doubles entrées. 70
3.11.1 - Algorithme LDEQUE. 71
3.11.2 - Algorithme L2QUE 73
3.12 - Listes partitionnées - Fonction Seuil 73
3.12.1 - Algorithme de partionnement. 74
3.12.2 - Fonctions avec Seuil 77

CHAPITRE 4 : Mise en oeuvre et expérimentations numériques 81

4.1 - Mise en oeuvre informatique d'un algorithme du chemin le plus court. 82
4.2 - Implantation en langage C. 87
4.3 - Algorithme LQUEUE, LDEQUE et L2QUE 91
4.4 - Algorithme LTHRS 97
4.5 - Algorithme SDKSTR. 101
4.6 - Algorithme SHEAP 103
4.7 - Tri par compartiments : algorithmes SDIAL et SRHEAP. 106
4.7.1 - Tri par compartiments. 106
4.7.2 - Algorithme SDIAL 112
4.7.3 - Algorithme SRHEAP. 115
4.8 - Résultats numériques 120
4.8.1 - Réseau complet 123
4.8.2 - Réseau aléatoire 127
4.8.3 - Réseau grille. 131
4.8.4 - Conclusions. 135

CHAPITRE 5 : Chemins les plus courts entre tous les couples de sommets. 138

5.1 - Transformation des longueurs des arcs. 139
5.2 - Concepts fondamentaux. 142
5.3 - Résultats fondamentaux 144
5.4 - Exemple. 149
5.5 - Algorithme de réoptimisation duale 152
5.6 - Algorithme DUAL. 156
5.7 - Algorithme LSM 160
5.8 - Expérimentations numériques. 163

CHAPITRE 6 : Structure des données 167

6.1 - Organisation logique 168
6.2 - Structure de données utilisées par les programmes. 173
6.3 - Banque de données. 186
6.4 - Conclusions. 192

Page

CHAPITRE 7 : Interface graphique 194
7.1 - Structure d'un module graphique. 195
7.2 - Les écrans 197
7.3 - La question technologique. 201
7.4 - Le langage de programmation. 202
7.5 - Système graphique de base. 204
7.5.1 - Concepts de base 205
7.5.2 - Structure logique de BGC 210
7.6 - Utilisation de la couleur. 216
7.7 - Représentation et affichage. 218
7.7.1 - Les éléments à afficher. 218
7.7.2 - Affichage du réseau physique 222
7.7.3 - Analyse par élément. 226
7.7.4 - Représentation "chemin". 234
7.8 - TACTIC du point de vue d'un usager 242

CONCLUSION 253

BIBLIOGRAPHIE. 257

REMERCIEMENTS. 263

LISTE DES TABLEAUX

Page

Tableau 4.1 : Caractéristiques de l'ordinateur SUN 3/180 utilisé au Centre de recherche sur les transports 122

Tableau 4.2 : Réseau complet, implantations traditionnelles. 124

Tableau 4.3 : Réseau complet, implantations avec des pointeurs 125

Tableau 4.4 : Réseau aléatoire, implantations traditionnelles. 128

Tableau 4.5 : Réseau aléatoire, implantations avec des pointeurs . . . 129

Tableau 4.6 : Réseau grille, implantations traditionnelles 132

Tableau 4.7 : Réseau grille, implantations avec des pointeurs. 133

Tableau 4.8 : Effet de lmax. 134

Tableau 5.1 : Réseau complet 164

Tableau 5.2 : Réseau aléatoire 164

Tableau 5.3 : Réseau grille. 165

LISTE DES FIGURES

Page

Figure 1.1 : Exemple de réseau de service et de quelques itinéraires 9

Figure 1.2 : Opérations et agrégations en transport des marchandises. 11

Figure 3.1 : Un 1-graphe 27

Figure 3.2 : Un chemin comprenant un circuit 31

Figure 3.3 : Un arbre partiellement ordonné. Les nombres dans les noeuds sont les clés 55

Figure 3.4 : Un tas. 56

Figure 3.5 : Organisation des données dans un schéma d'adressage dispersé. 57

Figure 3.6 : Arbre Rheap (lmax = 6). 60

Figure 3.7 : Recherche en largeur d'abord. 69

Figure 3.8 : Une pile. 70

Figure 3.9 : Structure de données utilisée par Pape (1974) 72

Figure 3.10: Double-file utilisée par Pallottino 73

Figure 4.1 : G représenté par la liste des successeurs. Lorsque les arcs sont munis de longueurs, celles-ci apparaissent dans les cellules des listes chaînées 83

Figure 4.2 : G représenté en machine avec les tableaux A[], ND[] et LNGTH[]. 84

Figure 4.3 : Une liste simplement chaînée implantée avec le tableau Q[]. 92

Figure 4.4 : Déroulement de la procédure QOUT. 100

Figure 4.5 : Représentation d'un tas 104

Figure 4.6 : Schéma d'une structure de compartiments implantée avec des listes doublement chaînées 107

Page

Figure 4.7 : Réseau complet : lmax=10 000, n=175 126

Figure 4.8 : Pourcentage d'amélioration des temps d'exécution de la figure 4.7 126

Figure 4.9 : Réseau alétoire : lmax=10 000, n=1000, m=10 000 130

Figure 4.10: Pourcentage d'amélioration des temps d'exécution à la figure 4.9. 130

Figure 4.11: Réseau grille, lmax=10 000, p=50, q=50. 134

Figure 4.12: Pourcentage d'amélioration des temps d'exécution de la figure 4.11. 135

Figure 5.1 : $T'_s = T^*_r$ 145

Figure 5.2 : $R_{s,u} = \{s,2,4,5\}$ 147

Figure 5.3 : T^*_r (r=1). 150

Figure 5.4 : T'_s (s=2). 150

Figure 5.5 : T'_s après un pivot 151

Figure 5.6 : Une itération de l'algorithme DUAL. 158

Figure 6.1 : Hiérarchie entre les divers éléments manipulés par l'outil TACTIC. 170

Figure 6.2 : Structure en cascade représentant la hiérarchie entre les réseaux. 172

Figure 6.3 : Un réseau et sa représentation par listes d'adjacence 173

Figure 6.4 : Représentation par multiliste du réseau de service (se référer au réseau physique de la figure 6.3). 175

Figure 6.5 : Représentation des routes 179

Figure 6.6 : Structure de multiliste pour représenter les itinéraires 180

Figure 6.7 : Une liste chaînée simple. 184

Figure 6.8 : Exemple de l'élimination du lien (2,4) quand le réseau est stocké dans des tables séquentielles 191

Page

Figure 7.1 : Structure des modules graphiques. 196

Figure 7.2 : Exemple d'une sortie graphique en format dessin 199

Figure 7.3 : Écran de type dessin. 200

Figure 7.4 : Schéma conceptuel d'un système graphique. 206

Figure 7.5 : Syntaxe des directives du fichier des dessins 208

Figure 7.6 : Structure de BGC. 211

Figure 7.7 : Quelques dessins obtenus à l'aide des primitives graphiques BGC. 217

Figure 7.8 : Réseau physique avec quelques caractéristiques physiques. 223

Figure 7.9 : Réseau physique avec flux total de marchandises sur les liens 225

Figure 7.10 : Représentation du flux sur un lien. 226

Figure 7.11a: Réseau physique avec flux total de marchandises aux noeuds. 227

Figure 7.11b: Une mesure pour un attribut, par lien, en histogramme 228

Figure 7.12 : Une mesure pour un attribut, par lien, en couleur 229

Figure 7.13 : Deux mesures pour un attribut, par lien, en histogramme à trois dimensions. 230

Figure 7.14 : Analyse par lien - deux mesures 231

Figure 7.15 : Analyse par lien - deux mesures - autre représentation. 232

Figure 7.16 : Analyse par noeud - deux mesures. 233

Figure 7.17 : Quelques services sur le réseau physique. 235

Figure 7.18 : Quelques services sur un même écran 237

Figure 7.19 : "Mauvaise" composition de l'image sur un terminal qui ne permet pas l'effaçage sélectif 238

Figure 7.20 : Flux de marchandises sur les arcs d'un service. 239

Page

Figure 7.21: Un marché et quatre itinéraires parallèles. 240

Figure 7.22: Un marché et quatre itinéraires non parallèles. 241

Figure 7.23: Représentation des chemins les plus courts. 244

Figure 7.24: Représentation conceptuelle d'un service. 247

INTRODUCTION

Le transport des marchandises occupe une place importante dans l'économie et ce à tous les niveaux : régional, national et international. C'est une activité complexe qui fait intervenir plusieurs niveaux de décision, de nombreux participants aux objectifs différents et souvent contradictoires et une quantité importante de ressources humaines et matérielles. Dans un tel contexte, la recherche opérationnelle et l'informatique peuvent être d'une aide considérable en offrant des outils qui facilitent et améliorent les nombreuses activités de planification.

La planification tactique et stratégique des opérations des transporteurs interurbains de marchandises, qui tout à la fois offrent des services de transport et contrôlent le cheminement des marchandises à travers leurs réseaux de service, constitue un problème, des plus importants pour le système de transport des marchandises. Dans ce contexte, l'application de tels outils s'avère très profitable. Le transport ferroviaire des marchandises, où les modes de transport correspondent aux différents services de train (normaux, directs, rapides, etc.), le transport intercité de lots brisés, où les divers types et combinaisons de remorques ainsi que le TOFC ("truck-on-flat-car") forment les modes de transport, les systèmes de transport multimodes de conteneurs, etc., sont des exemples de tels systèmes.

Récemment, Crainic et Rousseau (1986) et Crainic (1987) ont développé une approche de modélisation globale, basée sur des concepts d'optimisation des réseaux pour le problème de la planification tactique et stratégique de tels systèmes. Cette approche a été utilisée avec succès pour résoudre deux types de problèmes complexes. Le premier est un problème de transport par rail dont les données proviennent de réseaux canadiens (Crainic, 1984; Crainic et Rousseau, 1986) et français (Crainic et Nicolle,

1986). L'autre concerne le transport multimode des lots brisés par des compagnies de camionnage (Roy, 1984; Delorme et al., 1987; Roy et Delorme, 1989).

Le modèle tient compte des principaux aspects du processus de décision au niveau tactique de planification et considère les compromis à réaliser entre des critères de décision souvent contradictoires. Cette méthodologie permet d'améliorer le processus de planification par la génération et l'évaluation systématique et efficace d'un grand nombre de stratégies globales d'opération.

D'autre part, la complexité des composantes d'un système de transport interurbain des marchandises et leur haut degré de leurs interactions, la grande quantité des informations impliquées, ainsi que l'existence de règles et de critères difficilement modélisables compliquent la tâche de l'élaboration et de l'analyse d'un plan de transport. Aussi, notons que le problème de la planification tactique et stratégique d'un tel système possède un nombre élevé de solutions possibles avec sensiblement le même coût total. Dans ce contexte, il est intéressant de fournir au planificateur un environnement de travail interactif-graphique pour l'assister dans la conception d'un plan de transport. Avec un tel logiciel, le planificateur peut choisir parmi plusieurs solutions celle qui lui semble la plus souhaitable, pour ensuite la manipuler afin qu'elle respecte les règles et critères non inclus dans le modèle. Il peut également visualiser, analyser et comparer l'impact sur la forme, les coûts et qualité des opérations, de diverses décisions stratégiques, telles l'ouverture, la fermeture ou la modification d'installation, la fusion d'entreprises, etc.

Présentement, notre équipe travaille à la conception d'un modèle et d'un outil intégré d'aide à la planification tactique et stratégique du transport des marchandises (Crainic et Roy, 1988). L'outil, que nous appelons TACTIC, se compose d'un noyau méthodologique, d'une interface interactive-graphique ainsi que d'une banque de données.

L'objectif général de ce programme de recherche s'inscrit dans le cadre du projet TACTIC et il vise à intégrer les deux aspects de la programmation mathématique et informatique du problème. Il a pour but d'étudier et de mettre au point un prototype d'un système interactif-graphique, intégrant un modèle et des algorithmes de recherche opérationnelle, pour la planification tactique et stratégique du transport inter-urbain des marchandises. Nous ne prétendons pas, ici, à produire un logiciel complet, c'est là une tâche qui dépasse les objectifs d'un mémoire de maîtrise. Nous visons plutôt à établir les conditions pour qu'un tel logiciel soit opérationnel et à créer les fondations pour les travaux futurs.

Nous avons donc commencé la mise au point du prototype en concentrant d'abord nos efforts à l'analyse de la structure du modèle et des algorithmes de recherche opérationnelle. Plus précisément, nous avons abordé les algorithmes du chemin le plus court. Nous avons étudié les performances des nouveaux algorithmes par rapport à celui déjà utilisé pour l'optimisation de l'acheminement des marchandises à travers le réseau de services. Nous avons aussi examiné les algorithmes de réoptimisation pour la détermination des chemins les plus courts entre tous les noeuds du réseau.

Parallèlement, une partie importante de ce projet de recherche fut consacrée à l'étude des problèmes particuliers de représentation et d'affichage des éléments intervenant lors du procœssus de planification des

opérations de transport des marchandises. Nous avons implanté une bibliothèque de fonctions graphiques en langage C et nous discutons des solutions que nous apportons aux difficultés de représentation et d'affichage des éléments du modèle.

Voici maintenant, les grandes lignes de ce mémoire.

Au premier chapitre, nous révisons les principales étapes de l'élaboration d'un plan de transport. Nous détaillons les éléments du problème de planification tactique du transport des marchandises. De plus, nous y indiquons les principes de développement du prototype interactif-graphique TACTIC.

Par la suite, au deuxième chapitre nous introduisons des éléments de programmation mathématique, basés sur des concepts d'optimisation des réseaux, pour la résolution des problèmes du niveau tactique de la planification. Nous examinons le modèle mathématique et nous discutons des difficultés à le résoudre. Nous présentons une procédure de résolution basée sur des techniques de génération de colonnes. Aussi, nous étudions la place des problèmes du chemin le plus court rencontrés dans cet algorithme.

Au troisième chapitre, nous présentons une revue exhaustive de la bibliographie concernant le problème du chemin le plus court. Pour ce faire, nous utilisons l'approche unificatrice proposée récemment par Gallo et Pallottino (1986). Aussi, nous insistons sur les nouveaux algorithmes introduits par Pallottino (1979), Glover et al. (1984, 1985a, 1985b) et Ahuja et al. (1988).

À notre connaissance, il n'existe aucun résultat d'analyse numérique pour l'algorithme SRHEAP dû à Ahuja et al. (1988). Au quatrième chapitre, nous pallions à cette lacune en rapportant plusieurs résultats expérimentaux. Nous donnons le détail des implantations en langage C de huit algorithmes du chemin le plus court. Aussi, nous discutons des impacts de la programmation en langage C sur l'efficacité de ces algorithmes.

Le cinquième chapitre contient la description de deux algorithmes de réoptimisation pour la résolution du problème du chemin le plus court entre tous les noeuds d'un réseau. Nous établissons que les deux procédures sont des adaptations de la méthode duale du simplexe. Nous concluons avec quelques résultats d'expérimentations numériques.

La structure de la banque de données est discutée au sixième chapitre, tandis qu'au septième chapitre (le dernier mais non pas le moindre) nous passons en revue les solutions que nous avons apportées aux problèmes d'affichage graphique. Aussi, nous montrons comment se présente TACTIC du point de vue d'un utilisateur, en donnant un aperçu général des modules développés et de leur fonctionnement.

Enfin, les dernières pages de ce mémoire sont consacrées aux remarques finales et à quelques suggestions concernant les développements futurs de l'outil TACTIC.

CHAPITRE 1

SITUATION DU PROBLÈME

Au début de ce chapitre, nous révisons les principales étapes nécessaires à l'élaboration d'un bon plan de transport interurbain des marchandises. Par la suite, nous décrivons les éléments du problème de la planification tactique d'un système de transport. Cette revue est également utile pour présenter et établir les termes du vocabulaire utilisé dans ce mémoire. Nous poursuivons en concluant à la nécessité de mettre au point notre propre système interactif-graphique adapté aux problèmes que nous traitons. Enfin, le chapitre se termine par une présentation de la méthodologie et des concepts de base utilisés pour le développement de l'outil TACTIC.

1.1 Présentation du problème

La structure de base de tout système de transport de marchandises consiste en un réseau complexe de terminus reliés entre eux par des liens physiques ou conceptuels. Sur ce réseau, les services offerts par le transporteur sont généralement assurés par une flotte de véhicules qui transportent la marchandise entre des noeuds (paires origine-destination) du réseau. Ces véhicules se déplacent parfois en suivant un horaire et un itinéraire préétablis par le transporteur. La marchandise est habituellement classifiée par type de commodité, selon sa valeur ou ses caractéristiques physiques. Dans les terminus, diverses opérations doivent être réalisées soit, par exemple, la classification et la consolidation des marchandises.

Le problème du transport des marchandises peut se voir comme étant une suite de décisions que doivent prendre les autorités dans le but de répondre à la demande de service de la façon la plus adéquate possible. Les décisions portent principalement sur l'acquisition et l'affectation des

ressources et des pièces d'équipement. Le niveau de détails dans l'analyse menant aux décisions varie suivant le niveau de planification où l'on se situe. Il y a trois niveaux de planification dont voici la description.

Au **niveau stratégique,** nous retrouvons des décisions qui impliquent une grande partie de l'organisation, qui ont des incidences financières importantes et qui ont une portée à long terme. Ces décisions sont reliées à la conception du système de transport, c'est-à-dire au choix du type des marchandises transportées, des territoires desservis, des niveaux de services offerts à la clientèle et à la configuration du réseau de terminus et dans certains cas des liens physiques (par exemple, voies ferrées) requis.

Le **niveau opérationnel** concerne la gestion à très court terme, souvent hebdomadaire ou quotidienne et parfois en temps réel des opérations. On se préoccupe davantage des détails relatifs aux terminus et à l'équipement utilisé et on se situe souvent à des niveaux hiérarchiques peu élevés dans l'organisation. Les décisions importantes prises à ce niveau sont l'élaboration des horaires de transport et du personnel, l'affectation de l'équipement, la répartition des véhicules sur le réseau de transport, l'allocation des ressources dans les terminus et la gestion des retours de l'équipement vide.

Finalement, le **niveau tactique** se situe quelque part entre la planification stratégique à long terme et la gestion opérationnelle à très court terme. Son but est une affectation et une utilisation rationnelle, efficace et économique des ressources existantes en vue d'offrir un service de qualité. Les principales décisions qui composent cet horizon de planification concernent le type des convois à utiliser, leurs routes et fréquences, la détermination des classes de trafic à desservir par les divers services, la distribution des tâches de consolidation et classification

entre les terminus, l'établissement des politiques de classification des véhicules et, éventuellement, de formation des convois dans les terminus du réseau.

L'objet de notre recherche concerne le processus de planification tactique. Dans l'approche développée par notre équipe (Delorme et al., 1986; Crainic et Roy, 1989), ce processus est vu comme un problème d'optimisation qui peut être modélisé et résolu à l'aide de techniques de programmation mathématique. C'est un problème de design d'un réseau multimodal de services de transport, où l'objectif visé est la minimisation d'une fonction de coût généralisé qui tient compte des opérations de transport, des délais et de la fiabilité du service, etc.

1.2 Éléments du problème

Les divers éléments qui apparaissent lors du processus de planification tactique d'un système de transport de marchandises peuvent être regroupés en trois catégories : réseau physique, réseau de service et trafic des marchandises (Crainic, 1982; Roy, 1984; Crainic et Roy, 1989).

Le réseau physique se compose d'un ensemble de noeuds représentant des terminus de classification/consolidation, des stations de cueillette/livraison et des points de jonction. Les arcs reliant ces noeuds représentent soit des liens physiques (voies ferrées, routes), soit conceptuels (lignes maritimes, aériennes, etc.).

Sur le réseau physique, se surimpose un réseau de service assurant le transport du fret entre les noeuds (terminus et stations) du réseau.

Chaque service est caractérisé, dans le réseau physique, par un noeud origine et un noeud destination, une route les reliant et, éventuellement, des arrêts intermédiaires dans certaines stations sur la route. Le service est composé d'un ou plusieurs arcs de service, chacun de ces derniers représentant un sous-chemin entre deux arrêts consécutifs. Dans l'exemple illustré à la figure 1.1, les services A_1 et A_2 ont un arc de

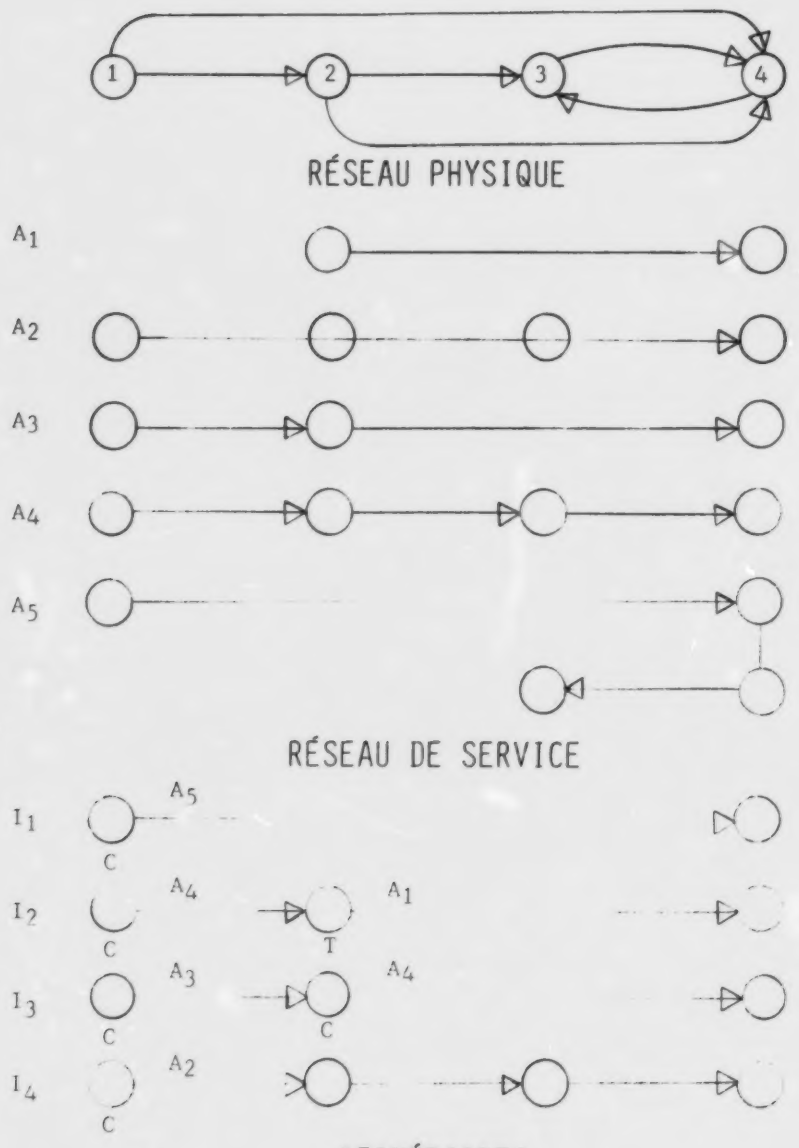


Figure 1.1 : Exemple de réseau de service et de quelques itinéraires

service , A_3 et A_5 en ont deux, tandis que A_4 est composé de trois arcs de service. Opérationnellement, chaque service est défini par son mode de transport (spécifiant sa vitesse, sa priorité, ses coûts d'opération et de délai) et par sa fréquence, spécifiant pour une période de planification le nombre de fois que le service est opéré. Lorsqu'au moins certains services de ce réseau multimode sont captifs de l'infrastructure (e.g. les divers types de trains du transport ferroviaire) on observe des phénomènes de congestion sur les liens qui en augmentent le délai de parcours.

La demande de transport, habituellement définie par marché, c'est-à-dire par origine-destination-classe de marchandise, est mesurée soit en tonnage, soit en nombre de véhicules. Les déplacements de marchandises n'étant pas individuels mais en formations plus ou moins agrégées, blocs, selon le mode de transport, la notion d'itinéraire est introduite pour les décrire. La figure 1.2 schématise les diverses opérations possibles et les niveaux d'agrégation qui leur correspondent. On trouve une description plus détaillée des opérations et agrégations des marchandises dans Crainic (1982). Chaque itinéaire pour un marché donné décrit un cheminement possible des marchandises du marché. Conceptuellement il peut se définir comme une suite de terminus où des opérations de classification et/ou consolidation sont effectuées sur la marchandise du marché et par les chemins de service utilisés pour se déplacer entre deux terminus adjacents de cette suite. La figure 1.1 montre quatre itinéraires possibles à partir du réseau de service illustré, où la lettre C indique une gare de classification-consolidation, tandis que la lettre T indique un transfert dans un chemin de service. Notons que plusieurs itinéraires peuvent être utilisés en même temps pour les mouvements de fret d'un même marché. Notons aussi que ces opérations d'agrégation physique du fret, tout en apportant des économies d'échelle aux coûts d'opération des systèmes de transport des marchandises, génèrent des phénomènes de congestion et des délais aux noeuds du réseau et augmentent la complexité du processus de planification.

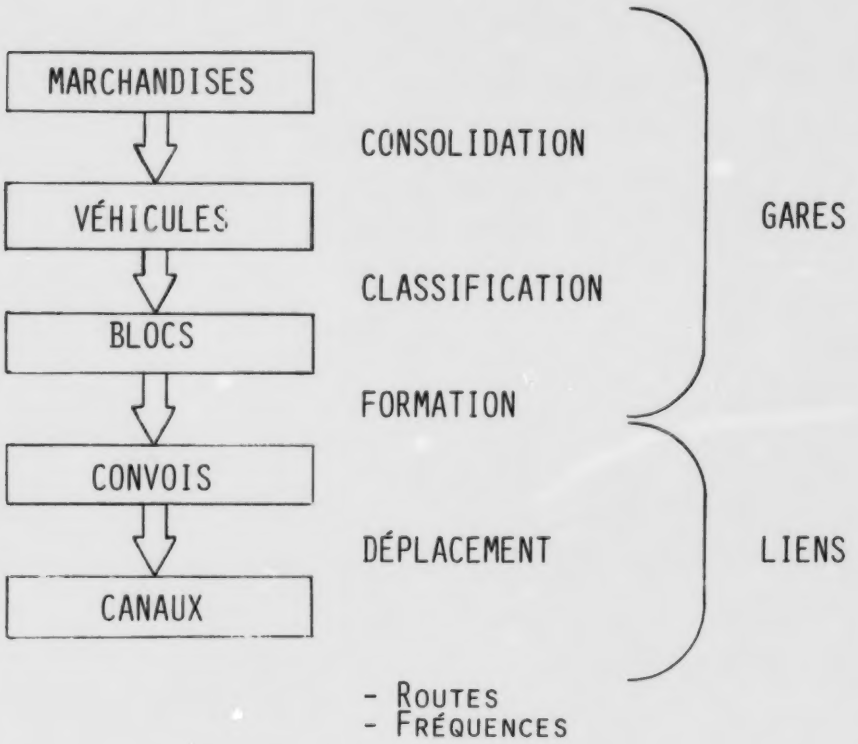


Figure 1.2 : Opérations et agrégations en transport des marchandises

Le résultat du processus de planification tactique d'un système de transport des marchandises est constitué d'un plan de transport spécifiant

- le design du réseau de services : la sélection des routes (noeud origine et noeud destination, route physique et arrêts intermédiaires) empruntés par les services de convois offerts à la clientèle. Détermination des caractéristiques de chaque service telles que son mode (par exemple, rapide, lent, etc.), sa priorité et sa fréquence;

- le trafic des marchandises : la spécification des services utilisés pour l'acheminement des marchandises de chaque marché ainsi que les opérations aux terminus de la route;

- la politique dans les terminus : l'établissement des règles générales, dans chaque terminus, de formation des blocs ainsi que la classification des véhicules dans ces blocs. L'établissement des politiques, pour chaque terminus, de formation des convois à partir des blocs de formation des convois à partir des blocs. La distribution du niveau de travail de consolidation et classification entre les terminus;

et conciliant les coûts d'opération et les critères de service. Mentionnons que la génération et l'analyse d'un tel plan de transport constitue un moyen d'évaluation privilégié pour les scénarios développés lors de la planification stratégique de l'entreprise.

1.3 L'approche interactive-graphique

L'élaboration et l'analyse d'un bon plan de transport est un problème difficile car, non seulement chaque composante est un problème complexe, de grande taille et dont l'impact est global sur tout le réseau, mais les interactions entre les composantes sont également nombreuses, fortes et complexes. Des compromis doivent également être réalisés entre des critères de coût d'opération et de qualité-fiabilité de service. De plus, le problème possède un grand nombre de solutions équivalentes, dans le sens où plusieurs combinaisons configuration de service-acheminement du fret offrent sensiblement le même niveau de service ou, approximativement, le même coût. D'autres critères discriminants sont alors utilisés, critères difficilement modélisables à l'aide de méthodes traditionnelles de recherche opérationnelle.

Pour ces raisons, il est important pour le planificateur de disposer d'un environnement de travail interactif-graphique pour l'assister dans la conception, l'analyse et la comparaison de plans de transport. Avec un tel outil, le planificateur peut choisir parmi plusieurs solutions celle qui lui semble la plus souhaitable, pour ensuite la manipuler afin qu'elle respecte les règles et les critères non inclus dans le modèle. Il peut

également visualiser, analyser et comparer l'impact sur la forme, les coûts et qualité des opérations de diverses décisions stratégiques, telle l'ouverture, la fermeture ou la modification d'installations, la fusion d'entreprises, etc.

À notre connaissance, il n'existe aucun système interactif-graphique adapté aux besoins d'un instrument de planification tactique et stratégique des réseaux de transport interurbain des marchandises. Toutefois, il existe quelques outils traitant les modèles de transport de marchandises au niveau national. Parmi ceux-ci, les plus importants sont le "Princeton Transportation Network Model and Graphic Information System" (Kornhauser, 1984, 1985a, 1985b) et STAN (Florian et al., 1988; Florian et Crainic, 1989; Crainic, 1989). Le premier consiste en un ensemble important de programmes, banque de données et procédures interactives graphiques conçus pour l'analyse des flux unimodaux du réseau nord américain des autoroutes et des voies ferrées. Cependant ce système ne répond pas aux problèmes que nous traitons car les procédures d'optimisation ne sont pas très sophistiquées et il est difficilement adaptable à d'autres contextes continentaux. Pour sa part, STAN est un logiciel interactif-graphique mis au point pour la planification stratégique du transport des marchandises au niveau national et régional. Le noyau méthodologique de STAN comprend un cadre normatif de modélisation et des algorithmes de recherche opérationnelle récents. C'est un système flexible et il permet de comparer graphiquement des scénarios. Par contre, le noyau méthodologique de STAN ne permet pas la représentation et la manipulation des chemins. Pour cette raison, STAN ne s'adapte pas non plus aux problèmes que nous traitons.

D'autre part, il existe plusieurs systèmes interactifs-graphiques pour la planification des réseaux urbains des personnes. Mentionnons les logiciels IGTDS ("Interactive Graphic Transportation Design", General Motors, 1978), TNOP ("Transit Network Optimization System", Rapp et al.,

1976), le système VOLVO (Andreasson, 1976; Hasselström, 1981) et EMME/2 ("Equilibrium Multimodal, Multimodal Equilibrium", Spiess, 1984). Bien qu'il soit généralement possible d'utiliser de tels systèmes pour analyser les mouvements des marchandises au niveau stratégique national (voir, par exemple, Lutin et al., 1979; Florian et al., 1986 et Guélat et al., 1987), ils se prêtent difficilement aux besoins du niveau tactique de planification d'un transporteur interurbain des marchandises. En effet, lorsque nous comparons les deux problèmes, nous observons des différences importantes, tant dans le type que dans la quantité d'informations manipulées, les plus significatives étant engendrées par la complexité des opérations aux noeuds des réseaux et l'existence de plusieurs classes de marchandises, ce qui nécessite la conservation et la manipulation explicites d'un grand nombre de chemins.

La conclusion : mettre au point notre propre système interactif-graphique, incluant un modèle et des algorithmes de recherche opérationnelle, qui répond exactement aux problèmes de la planification tactique et stratégique des réseaux de transport interurbain des marchandises.

1.4 Principes de développement

L'objectif général que nous poursuivons consiste à mettre au point un ensemble d'outils informatiques-graphiques pour assister la planification des systèmes de transport interurbain des marchandises. plus précisément, notre but est de permettre à un planificateur de

- traiter le problème globalement, au niveau de tout le réseau;
- conformer le modèle à son problème en spécifiant des fonctions de coût et de délai;
- générer, évaluer et comparer efficacement un grand nombre de stratégies d'opération tactique ou des scénarios stratégiques;

- manipuler et transformer une solution jusqu'à ce qu'elle respecte toutes les règles non écrites de la planification.

Du point de vue de l'implantation, notre logiciel doit permettre de

- manipuler efficacement une grande quantité d'informations pour la préparation de scénarios, l'analyse des données ou résultats;
- entrer et valider interactivement les données;
- analyser et comparer des résultats à l'aide des outils interactifs-graphiques;
- faciliter l'interaction planificateur-outil.

Aussi, afin de faciliter sa diffusion future dans les entreprises, le logiciel doit être facilement transférable d'un environnement informatique à un autre. Cette description des charges, bien que non exhaustive, constitue la liste essentielle de nos besoins.

Notre outil informatique, que nous appelons TACTIC, se compose de trois éléments :

a) **Noyau méthodologique** : modèles et algorithmes de recherche opérationnelle pour la génération, l'évaluation et la sélection de stratégies d'opération, pour effectuer des analyses post-optimales, etc.

b) **Banque de données** : fichiers sur disque pour emmagasiner les réseaux physiques, réseaux de service, fonctions de coût et de délai, demande de transport multiproduit, acheminement du fret et autres résultats et des données auxiliaires (par exemple, contours politiques).

c) **Interface interactive-graphique** : procédures pour entrer, manipuler et comparer des données interactivement et graphiquement.

Présentement, l'étude du noyau méthodologique est presque terminée. Rappelons les travaux importants de Crainic (1982), Crainic et Nicolle (1986), Crainic (1987) et Roy (1984). Nous disposons aussi de plusieurs primitives graphiques (Spiess, 1984 et Mondou et al. (1987) et nous avons déjà mené des expérimentations graphiques (Crainic et Mondou, 1986).

Dans ce mémoire, nous abordons principalement les points a) et c). Notre contribution à la partie méthodologique consiste à l'analyse détaillée des nouveaux algorithmes du chemin le plus court et des algorithmes de réoptimisation. Nous étudions leurs performances par rapport à celui déjà utilisé pour l'optimisation de l'acheminement des marchandises à travers le réseau de services. Nous menons les expérimentations numériques en langage C. Du point de vue des procédures graphiques, nous apportons plusieurs nouvelles solutions et nous avons mis en oeuvre plusieurs programmes et structures de données.

CHAPITRE 2

TACTIC : ÉLEMENTS DE PROGRAMMATION MATHÉMATIQUE

Nous rappelons que le but du processus de la planification tactique est une affectation et une utilisation rationnelle et efficace des ressources existantes pour améliorer la performance globale du système de transport. Les problèmes et les décisions considérés à ce niveau de planification ont des effets sur tout le réseau et sont fortement interreliés, tant dans leurs aspects économiques que dans leurs aspects spatio-temporels. Ainsi, pour pouvoir évaluer pleinement leur impact sur l'état et le fonctionnement du système, il faut traiter le problème globalement, au niveau de tout le réseau, et considérer non seulement les coûts d'opération mais aussi une mesure de la qualité du service offert (généralement, les délais imposés au trafic), ainsi que les relations et compromis à réaliser entre ces critères.

Pour ce faire, nous traitons le problème de la planification tactique des opérations du transport interurbain des marchandises à l'aide de techniques de programmation mathématique. Dans ce chapitre, nous décrivons une approche générale de modélisation qui intègre les diverses composantes, relations et compromis de ce problème de planification. Nous examinons le modèle mathématique et nous discutons des difficultés à le résoudre. Aussi, nous présentons une procédure de résolution basée sur des techniques de génération de colonnes. En dernier lieu, nous étudions la place des problèmes du chemin le plus court rencontrés dans cet algorithme.

2.1 Modélisation

Voici maintenant une brève présentation de la méthodologie que nous employons. Les détails peuvent être trouvés dans Crainic et Rousseau (1986) et Crainic (1987).

Le réseau physique est représenté par G=(N,L). Il est composé de l'ensemble N des noeuds représentant les jonctions, les stations et les terminus du réseau et de l'ensemble L des liens reliant ces noeuds. Nous identifions l'ensemble des terminus du réseau par Y, Y⊂N.

Sur le réseau physique, la demande de transport est spécifiée en termes de nombre de véhicules ou de tonnage d'un certain type de marchandise, c, à être transportée d'un terminus origine o, o∈Y à un terminus destination d, d∈Y. Par la suite, nous référons la demande de transport entre deux terminus du réseau à un marché m=(o,d,c) ayant une demande positive $g^m = g^m_{od}$ véhicules à transporter. L'ensemble M contient tous les marchés.

La structure des services, S, spécifie les services de convois offerts pour satisfaire la demande de transport. Chaque service, $s_t \in S$, est caractérisé par le chemin suivi dans le réseau physique, par un type spécifiant le mode de transport, la vitesse, la priorité, etc., et par une fréquence.

La **fréquence** d'un service, f_t, est définie comme étant le nombre de convois opérés sur le service durant la période de planification. Les valeurs des fréquences sont déterminées par le modèle. Une fréquence nulle signifie que le service correspondant n'est pas offert.

Pour un marché donné, le trafic emprunte généralement un seul chemin dans le réseau physique, sauf quand pour des raisons économiques ou de congestion il se divise entre plusieurs chemins différents. Pour circuler sur ces chemins, les véhicules utilisent un ou une suite de services. Les véhicules sont classifiés au terminus origine, mais peuvent aussi être reclassifiés dans certains noeuds intermédiaires.

La notion d'**itinéraire** est alors introduite afin de décrire un cheminement possible des véhicules d'un marché donné. Un itinéraire est la

séquence des terminus où les véhicules sont classifiés/re-classifiés, ainsi que les services utilisés pour se déplacer entre ces noeuds de classification. L'ensemble I^m contient tous les itinéraires desservant un marché. La valeur du flux à faire circuler sur chaque itinéraire est aussi calculée par le modèle.

Nous mentionnons que les **fréquences** et les **itinéraires** sont les éléments de base de cette méthodologie. Lorsque les valeurs des fréquences sont fixées, cela détermine le design du réseau de services, le niveau de service offert à la clientèle et le domaine réalisable pour l'acheminement du fret. D'autre part, la sélection des meilleurs itinéraires pour chaque marché résoud le problème de la détermination des flux optimaux des marchandises dans le réseau physique. De plus, cela spécifie le niveau de travail et les politiques générales de classification/consolidation dans les terminus du réseau.

Enfin, nous formulons le problème de la planification tactique des opérations du transport interurbain des marchandises de la manière suivante:

$$\text{Minimiser} \quad \Psi(x,f) = \sum_{m \in M} \sum_{k \in I^m} Z_k^m(x,f) + \sum_{s_t \in S} Y_t(f) + P(x,f)$$

$$\text{sujet à} \quad \sum_{k \in I^m} x_k^m = g^m \qquad \text{pour tout } m \in M$$

$$x_k^m > 0 \qquad \text{pour tout } k \in I^m,\ m \in M$$

$$f_t > 0 \text{ , entier, pour tout } s_t \in S$$

où, x_k^m : quantité de flux du marché m circulant sur son itinéraire k;

f_t : fréquence du service s_t;

x : vecteur des flux;

f : vecteur des fréquences;

$Z_k^m(x,f)$: coût total (opération et délai) d'utilisation de l'itinéraire k pour acheminer du trafic du marché m; la somme représente le coût "variable" d'acheminement du flux x par le réseau de service opéré au niveau f;

$Y_t(f)$: coût total d'opération du service s_t; la somme représente le coût "fixe" d'offrir un réseau de service opéré au niveau f;

$P(x,f)$: termes spéciaux pour modéliser des relations spécifiques, telle la capacité des services, comme coûts de délai.

La fonction objectif représente le coût total du système : coûts d'acheminement du trafic plus les coûts relatifs à l'opération du système. C'est un coût généralisé dans le sens où tant les coûts d'opération que les coûts de temps sont considérés. Les termes de délai sont transformés en coûts via des coûts unitaires de délai définis par l'usager pour chaque marché, type de service ou de véhicule. Ces coûts sont généralement basés sur des valeurs de dépréciation et d'inventaire des biens et des véhicules, mais peuvent aussi être utilisés pour modéliser certaines caractéristiques de "temps", telles des priorités entre les marchés ou le caractère d'urgence de certaines livraisons. On trouve une excellente discussion sur le choix des fonctions de coût et de délai pour le cas du transport des marchandises par rail dans Crainic (1987) tandis que Roy (1984), et Delorme et al. (1987) discutent de l'application de ce modèle pour le camionnage de lots brisés.

2.2 Résolution

La formulation mathématique induit un problème de multiflux, multiproduit non linéaire mixte de grande taille (Crainic et Rousseau, 1986). Principalement, les difficultés rencontrées pour la résolution exacte de ce problème se situent au niveau de la fonction objectif et au niveau du domaine réalisable de certaines variables.

La fonction objectif est non convexe et non séparable, ni par marché, ni par service et elle inclut tous les liens entre ces éléments. L'absence de la convexité provient du fait que les fonctions de coût et de délai ne sont pas nécessairement convexes. Nous insistons aussi sur la présence des contraintes d'intégralité sur les fréquences des services. Ces variables sont restreintes au domaine {0,1,2,3,...}. Rappelons que cette contrainte est plus difficile à traiter que la contrainte d'intégralité 0-1. Finalement, à cause de la définition combinatoire des itinéraires, le nombre possible de ceux-ci est très élevé.

Étant donné ces difficultés, le problème est résolu par une procédure heuristique de décomposition et de génération de colonnes qui prend avantage de la structure des contraintes (Crainic et Rousseau, 1986):

- absence, au niveau des contraintes, de toute relation entre les marchés et les services;
- les contraintes de distribution du trafic pour chaque marché sont linéairement indépendantes et forment une matrice bloc-diagonale avec une contrainte par bloc.

Pour une description détaillée de l'algorithme et de ses performances, voir Crainic et Rousseau (1986). L'algorithme travaille alternativement sur les sous-problèmes suivants :

1. Design de réseau de service. Étant donné la distribution du trafic de chaque marché parmi ses itinéraires, modifier les fréquences de service pour améliorer (si possible; sinon arrêter) la valeur de l'objectif. Les fréquences à modifier sont déterminées par des calculs de différences finies.

2. Distribution du trafic. Compte tenu du niveau de service, établir une distribution optimale du trafic. Le problème est résolu en appliquant un algorithme de décomposition qui parcourt de façon cyclique l'ensemble

des marchés, chaque sous-problème étant résolu en maintenant fixes les valeurs des variables des autres marchés. Pour chaque sous-problème, un nouvel itinéraire est d'abord généré par des méthodes de chemin le plus court, suivant quoi, la distribution du trafic est réoptimisée par des algorithmes de descente.

Mentionnons que cette méthodologie, qui a été expérimentée avec succès (Crainic, 1984; Crainic et Nicolle, 1986; Crainic et Rousseau, 1986; Roy, 1984; Delorme et al., 1987), est très modulaire tant au niveau de la formulation qu'à celui de l'algorithme.

Cela permet, d'une part, de modifier, le cas échéant, les algorithmes suivant les modifications du modèle et, d'autre part, d'utiliser seulement une partie de l'algorithme pour effectuer des études spécifiques, telle l'optimisation du trafic d'un ensemble de marchés. Ceci est facilité par l'efficacité des procédures individuelles composant l'algorithme. Notons également que le problème que nous étudions ne possède pas une solution optimale unique. En fait, il y a un nombre relativement grand de solutions possibles avec sensiblement le même coût total. Dans ce contexte, l'interaction entre le planificateur et le modèle intégré dans un système informatique, prend une place importante dans le choix d'une solution parmi toutes celles dont les coûts totaux sont semblables, mais qu'il peut départager selon des critères non exprimés et souvent difficillement exprimables dans un modèle normatif.

2.3 Importance des chemins les plus courts

Nous rappelons qu'un itinéraire est défini par la séquence des terminus où les véhicules sont classifiés/reclassifiés ainsi que les services utilisés pour se déplacer entre les terminus de cette séquence. Cette définition, combinatoire dans son essence, a pour effet que le nombre d'itinéraires possibles est très élevé. Pour cette raison, une procédure de génération progressive de colonnes a été incluse dans l'algorithme. Ce processus peut être vu comme une variante du problème du chemin le plus

court (voir Crainic, 1982). Dans cette dernière section, nous décrivons le problème de la génération d'un nouvel itinéraire et nous insistons sur la place qu'occupent les algorithmes de chemin le plus court dans cette procédure.

Lorsque nous générons un nouvel itinéraire pour un marché donné, nous voulons obtenir une variable qui améliore la solution courante. Donc, nous cherchons à déterminer le "meilleur" itinéraire par rapport aux conditions existantes : le niveau de service (fréquences des services) et la congestion sur les liens et aux noeuds du réseau. Cette notion étant définie par rapport aux aspects quantitatifs de ces facteurs, le "meilleur" itinéraire est celui qui offre le coût total (délai et opération) par unité de flux le plus petit. Autrement dit, la meilleure variable (itinéraire) est celle qui a le plus petit coût (délai et opération) marginal pour la première unité de flux qui lui est affectée.

Dans ce contexte, le coût d'un itinéraire est défini comme étant la somme des coûts des chemins, dans le réseau de service, empruntées par les marchandises entre les terminus où il y a classification, plus le coût des opérations de classification. Le coût de classification (opération et délai) à un terminus se décompose en deux parties :

- un coût fixe qui dépend uniquement du niveau de travail à ce terminus (congestion, caractéristiques physiques, etc.);
- un coût variable qui dépend du prochain terminus où la marchandise est classifiée.

En premier lieu, nous constatons que dans le but de générer le meilleur itinéraire, le coût du chemin, suivi par le fret, dans le réseau de services entre deux terminus où il y a classification doit être le plus petit parmi les coûts de tous les chemins qui relient ces deux noeuds. Par conséquent, la première phase du processus de la génération d'un itinéraire consiste à déterminer les chemins les plus courts, dans le réseau de

service, entre tous les terminus. Pour ce problème, les "longueurs" associées aux arcs de ce réseau sont des coûts unitaires par unité de flux composé d'un coût de délai et d'opération sur les liens et d'un coût de pénalité si la capacité est dépassée. En plus de cela, nous devons considérer un coût aux noeuds qui est soit le coût du délai de fréquence si le segment précédent provient d'un service différent (ou si c'est le terminus origine de l'itinéraire) soit du coût du délai de halte dans le cas contraire.

Ce problème est un problème de chemin le plus court entre tous les couples de sommets d'un graphe avec des pénalités de virage. Il se résoud de la même manière qu'un problème de chemin le plus court ordinaire (voir les chapitres 3 et 5) à la seule différence que les étiquettes sont affectées aux liens à la place des sommets. Une excellente discussion sur ce sujet se trouve dans Kirby et Potts (1969). Actuellement, ce problème est résolu par un algorithme utilisant un tas ("binary heap") qui est appliqué tour à tour à chacun des noeuds du réseau.

Enfin, nous obtenons un nouvel itinéraire pour le marché (o,d,c) en résolvant une variante du problème du chemin le plus court entre les terminus origine et destination du marché dans le graphe $G_I=(y,A_I)$ où y est le même ensemble des terminus défini à la section 2.1 et où les arcs $(i,j)\in A_I$ représentent les meilleurs chemins trouvés à l'étape précédente.

L'algorithme employé pour résoudre ce problème est un algorithme de recherche du minimum d'abord (voir chapitre 3) où la règle d'insertion des sommets dans la file de priorité est légèrement modifiée afin de tenir compte du coût variable qui dépend du prochain terminus où la marchandise est classifiée. L'algorithme termine quand le noeud destination de l'itinéraire reçoit son étiquette permanente. Comme précédemment, la file de priorité utilisée par Crainic (1982) est un tas.

Mentionnons que pour les deux problèmes de chemin le plus court, les coûts sur les arcs et les coûts aux noeuds ne sont jamais négatifs. De plus, des techniques récentes (Sedgewick et Vitter, 1984; Bovet, 1986) ont été mises au point pour accélérer les algorithmes de recherche du minimum d'abord quand ils sont appliquées pour la recherche d'un chemin le plus court allant d'un sommet à un autre, comme c'est le cas pour le dernier problème. Nous étudions ces techniques à la section 3.8.5.

Nous concluons ce chapitre en soulignant que le temps CPU nécessaire à la résolution des problèmes de chemin le plus court lors des expérimentations numériques menées par Crainic (1982) représentent environ 35 % du temps total pris pour la résolution des problèmes tests. Pour cette raison, nous croyons que l'étude des nouveaux algorithmes de chemin le plus court et leur implantation efficace en langage C peut contribuer à améliorer les performances des algorithmes de recherche opérationnelle employés dans TACTIC.

CHAPITRE 3

CHEMIN LE PLUS COURT : REVUE DE LA BIBLIOGRAPHIE

Les problèmes de cheminement figurent parmi les problèmes les plus anciens de la théorie des graphes. Parmi ceux-ci, le problème du chemin le plus court est le plus typique. Il se rencontre soit directement, soit comme sous-problème dans de nombreuses applications. Par exemple, nous avons vu au chapitre précédent l'importance prise par ce problème pour la résolution du modèle de la planification tactique du transport interurbain des marchandises. Malgré sa simplicité, le problème du chemin le plus court attire l'attention de nombreux chercheurs et de nouvelles contributions continuent d'apparaître dans la littérature scientifique.

Dans ce chapitre, nous présentons une revue exhaustive de la bibliographie et nous insistons sur les travaux récents. En premier lieu, nous définissons les termes utilisés au cours de ce chapitre et nous énonçons formellement le problème du chemin le plus court. Nous stipulons des hypothèses et nous montrons des conditions d'optimalité pour ce problème. Par la suite, nous formulons le problème du chemin le plus court comme un problème de détermination d'un flux à coût minimum dans un réseau. Cette formulation permet de déduire une première méthode de résolution basée sur l'algorithme du simplexe. Cela est aussi utile pour la description des algorithmes de réoptimisation que nous faisons au chapitre cinq. Enfin, nous présentons une approche unificatrice pour la description des algorithmes du chemin le plus court. Dans cette approche, ces algorithmes proviennent d'une procédure générale où la seule imprécision est le choix de la structure de données employée pour maintenir la liste des noeuds candidats. Nous terminons par la description de huit algorithmes du chemin le plus court dont les derniers algorithmes dus à Pallottino (1979), Glover et al. (1985) et Ahuja et al. (1988).

3.1 Notations-Définitions

Soit G=(N,U) un **graphe orienté.** N est l'ensemble des sommets ou des noeuds et U celui des arcs. Si n=|N| est le nombre de sommets, nous disons que le graphe G est fini d'ordre n, car nous supposons qu'il y a un nombre fini de sommets, numérotés i=1,2,...,n. Les éléments u∈U sont des couples ordonnés de sommets appelés des arcs. Si u=(i,j) est un arc du graphe G, i est l'extrémité initiale de u et j est l'extrémité terminale. Graphiquement, les sommets peuvent être représentés par des points et u=(i,j) est figuré par une flèche joignant les deux points i et j, j étant la pointe de la flèche. Un arc u=(i,i) dont les extrémités coïncident est appelé une boucle. Un **p-graphe** est un graphe dans lequel il n'existe jamais plus de p arcs de la forme (i,j) entre deux sommets quelconques i et j, pris dans cet ordre. En particulier, un 1- graphe est un graphe tel qu'il n'existe jamais plus d'un arc de la forme (i,j) pour tout i et pour tout j. Par la suite, nous considérons seulement les 1-graphes. Notons qu'il est toujours possible de transformer un p-graphe en un 1-graphe en ajoutant des sommets artificiels pour briser les arcs parallèles. Si m=|U| est le nombre d'arcs, la densité d'un graphe est $\frac{m}{n}$. Un graphe est dense lorsque m est le l'ordre de n^2.

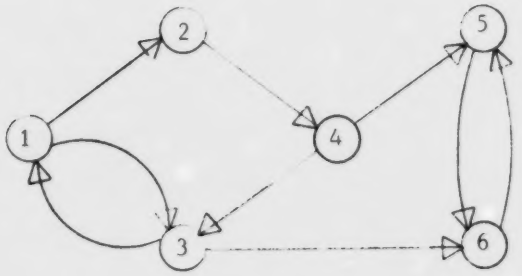


Figure 3.1 : Un 1-graphe

Une **chaîne** de cardinalité q est une séquence de q arcs $L=\{u_1,u_2,\ldots,u_q\}$ telle que chaque arc u_r de la séquence $(2<r<q-1)$ a une extrémité commune avec l'arc u_{i-1} et l'autre extrémité commune avec l'arc u_{i+1}. L'extrémité i de l'arc u_1 non adjacente à u_2 et l'extrémité j de u_q non adjacente à u_{q-1} sont appelées les extrémités de la chaîne L. Une chaîne est dite **élémentaire** si en la parcourant, on ne rencontre pas deux fois le même sommet. Un **cycle** est une chaîne dont les extrémités coïncident.

Un **chemin** de cardinalité q est une séquence de q arcs $P=\{u_1,u_2,\ldots,u_q\}$, avec

$$u_1=(i_0,i_1),\ u_2=(i_1,i_2),\ \ldots,\ u_q(i_{q-1},i_q)\ .$$

Autrement dit, un chemin est une chaîne dont tous les arcs sont orientés dans le même sens. Le sommet i_0 est l'extrémité initiale du chemin P et le sommet i_q est l'extrémité terminale du chemin P. Un chemin **élémentaire** est un chemin tel qu'en le parcourant, on ne rencontre pas deux fois le même sommet. Un **circuit** est un chemin dont les extrémités coïncident.

Par définition, le sommet j est un **successeur** du sommet i s'il existe un arc ayant i comme extrémité initiale et j comme extrémité terminale. L'ensemble des successeurs d'un sommet $i\in N$ est noté par Γ_i.

$$\Gamma_i = \{j \mid j\in N,\ (i,j)\in U\}$$

Réciproquement, j est un **prédécesseur** de i s'il existe un arc de la forme (j,i) dans G. L'ensemble des prédécesseurs du noeud i est noté par Γ_i^{-1}.

$$\Gamma_i^{-1} = \{j \mid j\in N,\ (j,i)\in U\}$$

L'ensemble de tous les sommets que l'on peut atteindre par un chemin à partir du sommet i est l'ensemble des **descendants** de i. Il est symbolisé par $\hat{\Gamma}_i$.

$$\hat{\Gamma}_i = \Gamma_i \cup \Gamma_i^2 \cup \ldots \cup \Gamma_i^{n-1}$$

où Γ_i^k représente l'ensemble des sommets que l'on peut atteindre à partir du sommet i par des chemins ayant exactement k arcs. De la même façon, $\hat{\Gamma}_i^{-1}$ représente l'ensemble des ancêtres de i. L'ensemble des arcs ayant comme origine le sommet i est noté par Λ_i ("forward star"),

$$\Lambda_i = \{(i,j) \mid (i,j) \in U, \text{ pour tout } j \in N\}$$

et l'ensemble des arcs ayant comme extrémité terminale le sommet i est représenté par Λ_i^{-1} ("backward star"),

$$\Lambda_i^{-1} = \{(j,i) \mid (j,i) \in U, \text{ pour tout } j \in N\} .$$

Il est clair que $U = \bigcup_{i \in N} \Lambda_i = \bigcup_{i \in N} \Lambda_i^{-1}$. De ceci, nous déduisons que le graphe peut être représenté par

$$G = \{(i, \Lambda_i),\ i=1,2,\ldots,n\} .$$

C'est la description "forward star" d'un graphe.

Le **sous-graphe** engendré par $A \subset N$ est le graphe G_A dont les sommets sont les éléments de A et dont les arcs sont les arcs de G ayant leurs deux extrémités dans A. Par exemple, si G est le graphe représentant les routes du Canada, celui qui représente les routes du Québec est un sous-graphe. Le **graphe partiel** engendré par $V \subset U$ est le graphe ayant le même ensemble N de sommets que G, et dont les arcs sont les arcs de V. Si G est le graphe représentant les routes du Canada, celui qui représente les routes à deux voies du Canada est un graphe partiel. Finalement, le **sous-graphe partiel** engendré par $A \subset N$ et $V \subset U$ est le graphe partiel de G_A engendré par V. Si G est le graphe représentant les routes du Canada, celui qui représente les routes à deux voies du Québec est un sous-graphe partiel.

Un **arbre** est un graphe connexe sans cycle. Un graphe est dit **connexe** si, pour tout couple de sommets i et j, il existe une chaîne joignant i et j. Un graphe T=(N,V) est une **arborescence** de racine r si T est un arbre dans lequel il existe, pour tout sommet $j \in N$, un chemin allant de r à j.

Enfin, un **réseau**, R=(N,U,L), est une structure formée d'un graphe G=(N,U) et d'un certain nombre de fonctions, L, définies sur ce graphe. Le graphe constitue la topologie du réseau. Les fonctions définies sur le graphe sont appelées les fonctions associées du réseau. Dans le cadre de ce travail, nous considérons uniquement la fonction de distance

$$\ell : U \to (-\infty, \infty) .$$

Le nombre ℓ_u est appelé la longueur de l'arc $u \in U$. Quand u désigne l'arc (i,j) $i,j \in N$, nous utilisons également la notation ℓ_{ij} pour désigner la longueur de l'arc u. Notons que, en toute généralité, les nombres ℓ_u peuvent prendre des valeurs positives, négatives ou nulles.

3.2 Problème du chemin le plus court

Le problème du chemin le plus court peut être posé de la façon suivante : nous supposons que le réseau R est fortement connexe. C'est-à-dire, pour chaque paire de sommets $i,j \in N$ il existe un chemin allant de i vers j reliant ces deux sommets. Nous dénotons par H_{ij} l'ensemble des chemins joignant le sommet i au sommet j. La longueur d'un chemin $h_{ij} \in H_{ij}$ est le nombre $d(h_{ij})$

$$d(h_{ij}) = \sum_{u \in h_{ij}} \ell_u$$

Un chemin le plus court de i à j, h^*_{ij}, est un chemin tel que

$$d(h^*_{ij}) = \min \{ d(h_{ij}) \text{ pour tout } h_{ij} \in H_{ij} \} .$$

Mentionnons que l'hypothèse de la forte connexité n'est pas contraignante. Dans le cas où cette condition n'est pas remplie, il suffit d'ajouter les arcs manquants munis de coût dont les valeurs sont très grandes.

3.3 Condition d'existence d'un chemin le plus court

Comme nous l'avons mentionné précédemment, en toute généralité, les longueurs des arcs peuvent être des nombres positifs, négatifs ou nuls. Considérons le chemin non-élémentaire h'_{ij} comprenant un circuit w.

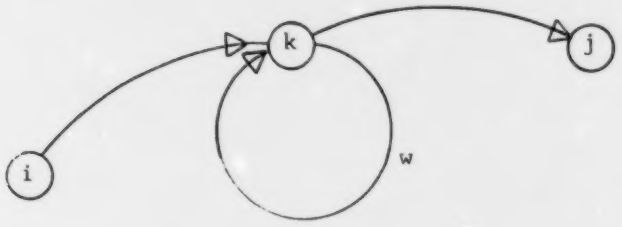


Figure 3.2 : Un chemin comprenant un circuit

De plus, notons par h_{ij} le chemin de i à j associé à h'_{ij}, mais n'empruntant pas le circuit w. La longueur du chemin h'_{ij} est donnée par

$$d(h'_{ij}) = d(h_{ij}) + d_w$$

où d_w est la longueur du circuit w.

Par conséquent, si la longueur du circuit w est négative, il n'existe pas de chemin le plus court entre i et j car, en l'empruntant sans cesse, une fois de plus, nous faisons décroître indéfiniment la longueur de

ce chemin. Un tel circuit porte le nom de **circuit absorbant.** Autrement, quand $d_w > 0$, nous avons $d(h_{ij}) < d(h'_{ij})$. Dans la recherche d'un chemin le plus court, nous pouvons donc nous restreindre aux chemins élémentaires entre i et j. Dans le cas où cette condition d'existence n'est pas remplie, nous posons le problème en termes de chemins élémentaires les plus courts. Puisque G est un graphe fini et fortement connexe, il existe toujours un chemin élémentaire de longueur finie entre tous les sommets. Toutefois, cette généralisation complique singulièrement le problème (voir Garey et Johnson, 1979) et, par la suite, nous supposons que le réseau R=(N,U,L) n'admet pas de circuit de longueur négative.

3.4 Principe d'optimalité d'un chemin le plus court

Le critère d'optimalité d'un chemin repose sur le fait que la longueur d'un chemin le plus court reliant les sommets i et j et passant par le sommet k vérifie nécessairement la relation

$$d(h^*_{ij}) = d(h^*_{ik}) + d(h^*_{kj}) \ .$$

En d'autres mots, le chemin le plus court reliant les sommets i et j et passant par le sommet k est l'union du chemin le plus court reliant les sommets i et k et du chemin le plus court reliant les sommets k et j. En effet, si l'un deux ne l'était pas, en lui substituant un chemin optimal, on définirait un chemin strictement préférable à h^*_{ij}.

Pour cette raison, il est intéressant de noter que trouver un chemin le plus court entre deux sommets r et t se ramène à déterminer un chemin le plus court entre un sommet r et chacun des autres sommets du graphe.

Dans ce qui suit, nous nous concentrons sur le problème de la recherche des chemins les plus courts issus d'un sommet r, parfois appelé la racine, vers tous les autres. Par définition, $r \in N$ est une racine du graphe G=(N,U) s'il existe un chemin joignant r à j, pour tout $j \in N$, $j \neq r$. Il résulte de la définition que, dans un graphe fortement connexe, tout sommet

est une racine. Nous remarquons aussi que si, au lieu d'imposer une extrémité initiale commune, nous souhaitons étudier le cas d'une extrémité terminale commune, il suffit d'inverser le sens de chacun des arcs du graphe pour se ramener au cas étudié ici.

Puisque le réseau R=(N,U,L) n'admet aucun circuit absorbant, nous avons $d(h^*_{ii})=0$ pour tout $i \in N$. Par ailleurs, considérons le chemin le plus court h^*_{rj} suivant :

$$\overset{0}{r} \rightarrow \overset{0}{\cdot} \rightarrow \overset{0}{\cdot} \rightarrow \overset{0}{\cdot} \rightarrow \overset{0}{i} \rightarrow \overset{0}{j}$$

La longueur de ce chemin est $d(h^*_{rj})$ et le dernier arc est (i,j). Par le critère d'optimalité, le sous-chemin allant de r à i est aussi un chemin le plus court et nous devons avoir

$$d(h^*_{rj}) = d(h^*_{ri}) + \ell_{ij} .$$

D'autre part, le sommet i doit être celui qui donne la plus petite valeur possible à la somme :

$$d(h^*_{ri}) + \ell_{ij} = \underset{k \neq j}{\text{Min}} \{ d(h_{rk}) + \ell_{kj} \} = d(h^*_{rj}) .$$

Il en résulte que les longueurs des chemins les plus courts issus du sommet r à chacun des $i \in N$ vérifient nécessairement les équations de Bellman (Lawler, 1976) :

$$\Pi_r = 0$$

$$\Pi_j = \underset{i \neq j}{\text{Min}} \{ \Pi_i + \ell_{ij} \}$$

Proposition 3.1

De plus, si le réseau ne possède aucun circuit dont la longueur est plus petite ou égale à zéro, alors les équations de Bellman possèdent une solution unique.

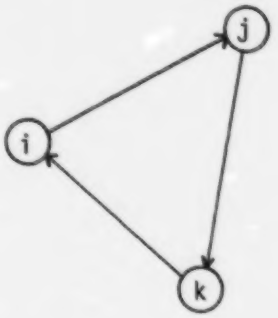
Preuve Le raisonnement qui conclut à la véracité de cette affirmation s'élabore comme suit :

Soit le vecteur $\Pi = (\Pi_1,\ldots,\Pi_n)$ une solution aux équations de Bellman. Nous allons montrer que $\Pi_i = d(h^*_{ri})$ pour tout i=1,2,...,n. Soit G' le graphe partiel de G engendré par $U' \subset U$:

$$(i,j) \in U' \text{ si } \Pi_j = \Pi_i + \ell_{ij}$$

G' n'a pas de circuit car si nous supposons le contraire nous arrivons à la contradiction suivante :

soit le circuit



Les équations relatives au critère d'optimalité pour les Π_i, Π_j et Π_k sont:

$$\Pi_i = \Pi_k + \ell_{ki}$$
$$\Pi_j = \Pi_i + \ell_{ij} = \Pi_k + \ell_{ki} + \ell_{ij}$$
$$\Pi_k = \Pi_j + \ell_{jk} = \Pi_k + \ell_{ki} + \ell_{ij} + \ell_{jk}$$

En remplaçant Π_i et Π_j par leurs expressions respectives nous obtenons :

$$0 = \ell_{ki} + \ell_{ij} + \ell_{jk}$$

Ceci contredit l'hypothèse que le réseau ne possède aucun circuit dont la longueur est plus petite ou égale à zéro. Donc, G' ne possède pas de circuit. Par construction, G' possède exactement n-1 arcs orientés dans le même sens, soit sortant du sommet r. Puisque G' est sans circuit, G' est une arborescence. Sous ces conditions Π_i représente la longueur d'un chemin de r à i.

Le reste de la preuve consiste à montrer que le vecteur $\Pi=(\Pi_1,\ldots,\Pi_n)$ est le vecteur des longueurs minimales. Pour ce faire, nous procédons aussi par contradiction. Supposons qu'il existe un indice j pour lequel $\Pi_j \neq \Pi_j^*$. Puisque G' est une arborescence, $\Pi_j > \Pi_j^*$. D'autre part, il existe au moins un indice i pour lequel $\Pi_i = \Pi_i^*$. Cela est vrai pour i=r. Conséquemment, il existe un couple (i,j) tel que $\Pi_i=\Pi_i^*$ et $\Pi_j>\Pi_j^*$. Ceci termine cette argumentation car

$$\Pi_j^* = \Pi_i^* + \ell_{ij} \qquad \text{(critère d'optimalité)}$$

$$\Pi_j > \Pi_j^* \rightarrow \Pi_j > \Pi_i^* + \ell_{ij}$$

$$\Pi_j > \Pi_i + \ell_{ij} \qquad (\text{car } \Pi_i=\Pi_i^*)$$

Mais Π est une solution aux équations de Bellman

$$\Pi_j = \underset{k\neq j}{\text{Min}} \{\Pi_k + \ell_{kj}\} > \Pi_i + \ell_{ij} \ .$$

■

Nous remarquons que les équations de Bellman sont non linéaires et récursives. Dans le cas général, ceci les rend très difficiles à résoudre directement. Par contre, lorsque le réseau R=(N,U,L) ne comporte pas de circuit, il existe une numérotation des sommets pour laquelle les équations de Bellman se résolvent trivialement (voir Gondran et Minoux, 1985). Les algorithmes proposés pour résoudre le problème du chemin le plus court sont parfois interprétés comme des algorithmes qui tentent de résoudre les équations de Bellman (Lawler, 1976).

Dans le but d'abréger la notation, par la suite nous employons Π_i^* pour désigner la valeur $d(h_{ri}^*)$.

3.5 Chemins optimaux en programmation linéaire

Nous pouvons modéliser le problème du chemin le plus court comme un problème de détermination d'un flux à coût minimum dans un réseau. Pour ce faire, le sommet r dispose de n-1 unités de flux et chacun des autres sommets du réseau doit en recevoir une. La longueur des arcs représente le

coût de transport par unité de flux et la fonction objectif consiste à minimiser le coût total de transport des n-1 unités. La variable de décision est x_{ij} et elle représente le flux sur chaque arc du réseau. Le modèle est le suivant :

Problème P1 $\quad \text{Min} \sum_{(i,j)\in U} \ell_{ij}\, x_{ij}$

s.a.

$$\sum_{(i,j)\in\Lambda_i} x_{ij} - \sum_{(j,i)\in\Lambda_i^{-1}} x_{ji} = \begin{cases} n-1 & \text{si } i=r \\ -1 & \text{sinon} \end{cases}$$

$$x_{ij} \geq 0 \quad \text{pour tout } (i,j)\in U$$

Puisqu'il n'y a pas de restriction sur la capacité des arcs, chaque unité de flux suit le chemin le plus court du sommet r jusqu'à sa destination i. Dans ce cas, nous pouvons montrer que les arcs $(i,j)\in U$ tels que $x^*_{ij} > 0$ forment une arborescence T_r dont la racine est le sommet r (Dantzig, 1963). Le problème linéaire dual correspondant est, si nous associons la variable duale $-\Pi_i$ à la contrainte i :

Problème P2 $\quad \text{Max} \sum_j (\Pi_j - \Pi_r)$

s.a.

$$\Pi_j - \Pi_i \leq \ell_{ij} \quad \text{pour tout } (i,j)\in U, \text{ pour tout } i\in N$$

Puisque les contraintes du problème primal comportent une équation redondante, nous pouvons fixer Π_r à zéro.

La matrice d'incidence sommets-arcs d'un réseau R=(N,U,L) est une matrice $A=(a_{iu})$, i=1,2,...,n, u=1,2,...,m à coefficients entiers 0, 1, -1 telle que chaque colonne correspond à un arc du réseau, et chaque ligne à un sommet de R. Si u=(i,j) est un élément de l'ensemble U, la colonne u a tous ses termes nuls, sauf :

$$a_{iu} = 1$$

$$a_{ju} = -1$$

Le rang d'une matrice d'incidence sommets-arcs est n-1 (voir Kennington et Helgason, 1980). Réécrivons le modèle linéaire P1 en notation matricielle :

$$\begin{aligned} &\text{Min} && \ell^T x \\ &\text{s.a.} && Ax = b \\ & && x \geq 0 \end{aligned}$$

où $\ell^T = (\ell_1,\ldots,\ell_m)$ est le vecteur des longueurs des arcs;

$b^T = (b_1,\ldots,b_n)$ où $b_i = \begin{cases} n-1 & \text{si } i=r \\ -1 & \text{sinon} \end{cases}$;

$x^T = (x_1,\ldots,x_m)$ est le vecteur des variables de décision.

Cette formulation permet de dériver l'expression arithmétique du coût réduit des arcs. Si $-\Pi_i$ est la variable duale associée à la $i^{\text{ième}}$ contrainte, $\bar{\ell}$, le vecteur des coûts réduits est

$$\bar{\ell}^T = \ell^T - (-\Pi)^T * A$$

$$\bar{\ell}_{ij} = \ell_{ij} + (\Pi_1,\ldots,\Pi_n)\, A_{\cdot u}$$

où $A_{\cdot u}$ est la $u^{\text{ième}}$ colonne de la matrice A.

Rappelons que la matrice d'incidence sommets-arcs contient exactement deux termes non nuls par colonne, dont l'un est +1 (a_{iu}), et l'autre -1 (a_{ju}). Par conséquent,

$$\bar{\ell}_{ij} = \ell_{ij} + \Pi_i - \Pi_j \quad \text{pour tout } (i,j)\in U .$$

Il est bien connu que le vecteur des multiplicateurs du simplexe associé à la base optimale du primal est aussi une solution optimale du problème dual. Dans le cas du programme linéaire P1, le théorème des écarts complé-

mentaires s'énonce de la manière suivante. Deux solutions (x^*,Π^*) du primal et du dual respectivement sont optimales si et seulement si

$$(\ell_{ij} + \Pi_i^* - \Pi_j^*) \cdot x_{ij}^* = 0 \quad \text{pour tout } (i,j)\in U .$$

Il est intéressant de noter que le théorème des écarts complémentaires pour le programme linéaire P1 est identique à la proposition 3.1 démontrée auparavant. En effet, la relation $\Pi_j^*=\Pi_i^* + \ell_{ij}$ est nécessairement vérifiée pour tout arc (i,j) tel que $x_{ij}^* > 0$. De plus, le graphe partiel déduit de R avec les arcs optimaux est une arborescence dont la racine est r.

Les variables duales optimales Π_i^* sont alors uniquement calculées en résolvant le système d'équations triangulaire

$$\Pi_r^* = 0$$

$$\Pi_j^* = \Pi_i^* + \ell_{ij} \quad \text{pour tout } (i,j) \text{ tel que } x_{ij}^* > 0$$

Les observations suivantes nous indiquent une procédure pour solutionner ce système.

L'interprétation économique des multiplicateurs optimaux est la longueur des chemins les plus courts issus du sommet r. Soit T_r^*, l'arborescence des chemins les plus courts formée avec les arcs (i,j) tels que $x_{ij}^* > 0$. Soit $P=(r,i_1,i_2,\ldots,i_k)$ un chemin partant de r jusqu'à un sommet i_k dans cette arborescence. Nous rappelons que ce chemin est uniquement constitué avec des arcs de base et que le coût réduit d'une variable de base est toujours zéro. Dans ces conditions, nous avons

$$\Pi_r^* - \Pi_{i_1}^* + \ell_{r,i_1} = 0 \rightarrow \Pi_{i_1}^* = \ell_{r,i_1}$$

$$\Pi_{i_1}^* - \Pi_{i_2}^* + \ell_{i_1,i_2} = 0 \rightarrow \Pi_{i_2}^* = \Pi_{i_1}^* + \ell_{i_1,i_2} = \ell_{r,i_1} + \ell_{i_1,i_2}$$

$$\vdots$$

$$\Pi_{i_{k-1}}^* - \Pi_{i_k}^* + \ell_{i_{k-1},i_k} = 0 \rightarrow \Pi_{i_k}^* = \Pi_{i_{k-1}}^* + \ell_{i_{k-1},i_k}$$

$$= \ell_{r,i_1} + \ell_{i_1,i_2} + \ldots + \ell_{i_{k-1},i_k}$$

Par la suite, il suffit d'appliquer ce raisonnement pour chaque sommet $i \in N$ pour vérifier que Π^*_i représente la longueur du chemin le plus court de la racine à ce sommet.

3.6 Méthode du simplexe

Quoiqu'il est possible d'appliquer directement la méthode du simplexe pour résoudre le programme linéaire primal ou dual, nous pouvons mettre en oeuvre plusieurs variantes de cette méthode qui prennent avantage de la structure de réseau du problème. D'une manière générale, la méthode du simplexe s'interprète algébriquement comme la détermination d'une suite de bases adjacentes $B_0, B_1, \ldots, B_q, \ldots$ et de solutions de base $x_1, \ldots, x_q, \ldots$ telles que la valeur de la fonction objectif décroît à chaque itération. L'algorithme se termine en un nombre fini d'itérations puisqu'il existe un nombre fini de points de base et que la décroissance stricte de la valeur de la fonction objectif interdit de passer deux fois par le même point. Or, comme l'ont montré Klee et Minty (1972), il existe des problèmes pour lesquels la méthode du simplexe nécessite l'examen de tous les points de base. Donc, la complexité de cet algorithme est exponentielle car il y a $c_n^m = \frac{m!}{(m-n)!n!}$ points de base. Cependant, il se révèle en moyenne, d'une très grande efficacité sur la plupart des problèmes pratiques (Dantzig, 1963). Géométriquement, la procédure s'interprète comme le cheminement d'un point extrême à un autre le long de la frontière d'un polytope convexe appelé simplexe. Quoiqu'en pratique, la méthode du simplexe fasse preuve d'un bon comportement du point de vue du nombre d'itérations, nous mentionnons que l'effort requis à chaque étape est considérable. En effet, les opérations élémentaires sur les lignes de la matrice A pour déterminer la forme canonique relative à la nouvelle base sont coûteuses. De plus, l'espace mémoire requis pour conserver la matrice de base est non négligeable. Toutefois, pour les problèmes de chemin le plus court, nous pouvons conserver la base B avec le vecteur p suivant :

$$p_j = \begin{cases} i & \text{si } (i,j)\in B \\ -1 & \text{si } j=r \end{cases}$$

Puisqu'une base B réalisable correspond à une arborescence T_r formée par les arcs de base du réseau, le changement de base correspond à l'échange d'un arc hors arborescence avec un arc de l'arborescence. Ceci nous permet d'effectuer les pivots de manière intégrée avec une structure de données adéquate. Une spécialisation de la méthode du simplexe pour résoudre le problème P1 qui met en oeuvre ces considérations est la suivante :

ALGORITHME 1

1. Initialisation

$k \leftarrow 0$ compteur d'itérations

$$\Pi_j \leftarrow \begin{cases} 0 & \text{si } j=r \\ \ell_{rj} & \text{si } (r,j)\in U,\ j\neq r \\ \text{infini} & \text{sinon} \end{cases}$$

$$p_j \leftarrow \begin{cases} -1 & \text{si } j=r \\ r & \text{sinon} \end{cases}$$

$T_r^k \leftarrow \{(r,j)\ j=1,2,\ldots,m\}$ certains arcs sont des arcs artificiels.

2. Test d'arrêt

Si pour tout $(i,j)\in U$ t.q. $\Pi_i - \Pi_j + \ell_{ij} \geq 0$
alors TERMINER

3. Sélection

Prendre un arc (i,j) quelconque pour lequel $\Pi_i - \Pi_j + \ell_{ij} < 0$

4. Pivot

$k \leftarrow k+1$

$T_r^k \leftarrow T_r^{k-1} - \{(p_j,j)\} \cup \{(i,j)\}$

$p_j \leftarrow i$

$\Pi_j \leftarrow \Pi_i + \ell_{ij}$

$\Pi_\ell \leftarrow \Pi_\ell - (\Pi_j - \Pi_i - \ell_{ij})$ pour tout $\ell \in T_j^k$

Aller à 1

■

Le principal avantage de l'algorithme 1 est qu'il travaille directement sur le réseau. Aussi, soulignons que pour un problème plus général de détermination d'un flux à coût minimum dans un réseau pour lequel les arcs ont des bornes supérieures (capacités), une base réalisable pour le programme linéaire associé correspond à un arbre et non pas nécessairement à une arborescence. C'est seulement pour le cas particulier du problème P1 qu'il y a une équivalence entre les bases réalisables et les arborescences. Nous référons le lecteur à Dantzig (1963) pour de plus amples détails concernant ce sujet. Pour ajouter à ce propos, soulignons l'excellent article de Brown, Graves et Bradley (1977) qui expose en détail une spécialisation de l'algorithme du simplexe pour résoudre un problème de détermination d'un flux à coût minimum de grande taille. D'autre part, Dial et al. (1979) rapportent la mise en oeuvre de plusieurs critères de sélection de l'arc d'entrée pour la résolution du problème du chemin le plus court avec la méthode du simplexe. On trouve une très bonne revue des procédures nécessaires à la manipulation des arbres pour les problèmes de réseau dans Pallottino et Storchi (1981, 1984), Fox (1978) et Kennington et Helgason (1980). En plus, ce dernier ouvrage expose en détail la théorie de la programmation linéaire pour les problèmes de réseau.

De manière informelle, l'algorithme 1 s'interprète comme la détermination d'une suite d'arborescences de racine r $T_r^o,\ldots,T_r^q\ldots$ Chacune des arborescences est associée à une solution de base réalisable pour le problème P1. Les variables duales représentent les longueurs des chemins

dans l'arborescence courante. À chaque pivot, une inégalité $\Pi_j - \Pi_i > \ell_{ij}$ est identifiée et l'arc améliorant (i,j) est introduit dans la nouvelle arborescence tandis que l'arc (p_j,j) en est retiré. Puisque nous supposons qu'il n'y a aucun circuit absorbant, ce changement d'arc permet de passer à une nouvelle arborescence. Les variables Π_k de la sous-arborescence T_j sont révisées à la baisse. Cette procédure crée une suite monotone non croissante pour chaque variable Π_i=1,2,...,n. La solution optimale est atteinte lorsque toutes les inégalités $\Pi_j - \Pi_i < \ell_{ij}$ sont satisfaites. À ce moment, l'arborescence correspondante est une arborescence optimale pour le problème du chemin le plus court. Le système d'inégalités précédent nous invite à considérer le vecteur Π comme un potentiel. Un prototype d'algorithme possible pour satisfaire ces conditions d'optimalité est alors le suivant (Roy, 1970):

ALGORITHME 2

1. Initialisation

$q \leftarrow 0$

$$p_j \leftarrow \begin{cases} 0 & \text{si } j=r \\ r & \text{sinon} \end{cases}$$

$$\pi_j^q \leftarrow \begin{cases} 0 & \text{si } j=r \\ \text{infini} & \text{sinon} \end{cases}$$

2. Sélection

Choisir un arc (i,j) tel que $\pi_j^q - \pi_i^q > \ell_{ij}$

3. Poursuite ou arrêt

S'il n'en n'existe pas, TERMINER

Sinon faire

début

$$\pi_j^{q+1} \leftarrow \pi_i^q + \ell_{ij}$$

$$\pi_k^{q+1} \leftarrow \pi_k^q \qquad \text{pour tout } k \in N,\ k \neq j$$

$p_j \leftarrow i$; $q \leftarrow q+1$

aller à 2

fin

En d'autres mots, l'algorithme débute avec un ensemble de nombres $\Pi_1^o,\ldots,\Pi_n^o$ dont nous sommes certains qu'ils majorent les longueurs optimales $\Pi_1^*,\ldots,\Pi_n^*$. Cela correspond à initialiser une arborescence de racine r avec des arcs artificiels. Ensuite nous modifions progressivement les valeurs de départ. La $(q+1)^{ième}$ modification s'effectue en choisissant, d'une façon qui peut être absolument quelconque, un arc (i,j) pour lequel :

$$\Pi_j^q - \Pi_i^q > \ell_{ij}$$

Si un tel arc existe nous faisons :

$$\Pi_j^{q+1} = \Pi_i^q + \ell_{ij} \qquad \text{(ajustement)}$$

$$\Pi_k^{q+1} = \Pi_k^q \qquad \text{pour tout } k \neq j$$

C'est un pivot, ou mise à jour de l'arbre. Contrairement à la méthode du simplexe, les variables duales du sous-arbre dont j est la racine ne sont pas mises à jour. Cependant, cela n'affecte pas la convergence de l'algorithme (voir plus loin). Lorsqu'après un certain nombre d'ajustements, P, aucun autre n'est plus possible, les nombres Π_i^p obtenus sont les longueurs optimales.

À chaque étape de l'algorithme, Π_i est un majorant de la longueur du chemin le plus court de r à i. D'autre part, le graphe est sans circuit absorbant, par conséquent, Π_i est borné inférieurement par la longueur du chemin le plus court de r à i. Puisque Π_i décroît à chaque étape et qu'il est borné inférieurement, il admet une limite et l'algorithme converge. Pour établir que cette limite est bien le résultat escompté, considérons un sommet i. Soit i_1 le dernier sommet utilisé pour diminuer Π_i. Donc, nous avons :

$$\Pi_i = \Pi_{i_1} + \ell_{i,i_1}$$

De la même façon, soit i_2 le dernier sommet utilisé pour diminuer Π_{i_1}. Nous obtenons :

$$\Pi_{i_1} = \Pi_{i_2} + \ell_{i_1,i_2}$$

Ainsi, nous définissons une suite de sommets deux à deux distincts avec r comme dernier sommet, car le graphe est d'ordre n fini. Soit donc i, i_1, ..., i_k, r cette suite. Nous avons alors

$$\Pi_{i_k} = \Pi_r + \ell_{i,i_k}$$

En sommant les équations précédentes, nous trouvons :

$$\Pi_i = \ell_{r,i_k} + \ell_{i_k,i_{k-1}} + \ldots + \ell_{i_2,i_1} + \ell_{i_1,i}$$

Puisque $\Pi_r = 0$, Π_i représente donc la longueur d'un chemin de r à i.

Pour terminer cette justification, remarquons que les nombres auxquels nous parvenons en fin de calcul sont solutions du système d'inégalités de la proposition 1 (équations de Bellman). Considérons un autre chemin de r à i, soit :

$$\mu = (r, t_h, t_{h-1}, \ldots, t_1, i)$$

Par conséquent, nous avons :

$$\Pi_i \leq \Pi_{t_1} + \ell_{t_1,i}$$

$$\Pi_{t_1} \leq \Pi_{t_2} + \ell_{t_2,t_1}$$

$$\vdots$$

$$\Pi_{t_h} \leq \Pi_r + \ell_{r,t_h}$$

C'est-à-dire :

$$\Pi_i \leq \ell_{r,t_h} + \ell_{t_h,t_{h+1}} + \ldots + \ell_{t_2,t_1} + \ell_{t_1,i} = \ell_\mu$$

Π_i est donc bien la longueur du chemin le plus court de r à i.

C'est à cette classe qu'appartiennent les premiers algorithmes proposés, en particulier par Ford (1956) et Moore (1959). Par définition, les nombres Π_i^0 sont des bornes supérieures aux Π_i^*. Il est facile d'établir par récurrence que cette propriété est conservée à chaque modification. C'est pour cela que les algorithmes qui appartiennent à cette catégorie sont dits des algorithmes qui procèdent par ajustements progressifs. De plus, les nombres $\Pi_1,\ldots,\Pi_n$ sont des étiquettes ou des marques. On parle aussi d'algorithme de marquage.

3.7 Approche unificatrice

Pratiquement, nous n'utilisons pas l'algorithme précédent mais une amélioration qui tient compte que le réseau peut être facilement donné par l'ensemble des successeurs de chaque sommet. Le principe est le suivant (Yen, 1970). Nous cherchons les arcs améliorants que parmi les sommets i dont l'étiquette Π_i a été précédemment modifié. Pour y parvenir, nous choisissons un sommet dont l'étiquette a été rendue meilleure et nous analysons les arcs sortant de ce sommet. Cela semble d'autant plus raisonnable de sélectionner les sommets à la place des arcs, car n est beaucoup plus petit que m. Récemment, Gallo et Pallottino (1986) ont établi que la recherche des chemins les plus courts repose sur un algorithme général basé sur l'exploration complète des arcs sortants d'un sommet. En effet, la plupart[1] des algorithmes proposés pour la recherche des chemins minimaux procèdent de la manière suivante. la méthode débute avec une arborescence T constituée du sommet r et des arcs artificiels (r,i), i≠r, dont les

---

[1] Les algorithmes par calcul matriciel sont exclus de ce modèle général.

longueurs sont très grandes. L'ensemble des sommets à explorer, Q, est initialisé avec le sommet r. Une itération de la procédure consiste à retirer d'une façon arbitraire un sommet de cet ensemble et à en balayer les arcs sortants. Les sommets améliorés par ces arcs sont placés dans Q tandis que les arcs améliorants remplaçent d'autres arcs dans T. Le processus termine quand l'ensemble des sommets à explorer est vide.

Gilsinn et Witzgall (1973) utilisent le terme "sequence list" pour désigner l'ensemble des sommets à explorer tandis que d'autres, Glover et al. (1985b), emploient l'expression "scan eligible (SE) node list". Pour notre part, nous employons la lettre majuscule Q pour identifier cette liste de sommets. Une liste est une collection d'éléments liés entre eux et dont la manipulation est régie par le type de la liste. Par exemple, les listes les plus connues sont les files et les piles. Le premier élément de la liste est la tête et le dernier est la queue. Les opérations permises sur une liste sont l'ajout et la suppression d'éléments.

Formellement, un prototype d'algorithme basé sur l'exploration complète des arcs sortants d'un sommet est le suivant.

ALGORITHME 3

1. Initialisation

$$p_j \leftarrow \begin{cases} 0 & \text{si } j=r \\ r & \text{sinon} \end{cases}$$

$$\Pi_j \leftarrow \begin{cases} 0 & \text{si } j=r \\ \text{infini} & \text{sinon} \end{cases}$$

$Q \leftarrow \{r\}$

2. Sélection, amélioration

Sélectionner un i quelconque dans Q
$Q \leftarrow Q - \{i\}$

Pour chaque $(i,j) \in A_i$ faire

{ si $\Pi_i - \Pi_j + \ell_{ij} < 0$ alors

début

$\Pi_j \leftarrow \Pi_i + \ell_{ij}$

$p_j \leftarrow i$

$Q \leftarrow Q \cup \{j\}$

fin

}

3. Arrêt

Si $Q \neq \emptyset$ aller à 2

Sinon STOP

À ce stade-ci, la seule imprécision qui persiste est la manière d'exécuter les opérations de sélection-suppression ($Q \leftarrow Q - \{i\}$) et d'insertion ($Q \leftarrow Q \cup \{j\}$) des sommets dans Q. Or, sans que cela soit connu, nous établissons que tout calcul qui se déroule de la sorte conduit bien aux longueurs des chemins minimaux en remarquant que l'algorithme 3 se ramène à appliquer l'algorithme 2 où le choix des arcs améliorants est prescrit par l'ordre des sommets dans Q[1]. Contrairement à la méthode du simplexe où à chaque itération le vecteur $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ représente les longueurs des chemins dans l'arborescence T courante. Ici, cette propriété est vérifiée seulement à la fin de l'algorithme. Entre-temps, les valeurs Π_i sont des majorants de ces longueurs. Shier et Witzgall (1981) utilisent le qualificatif "sharp" pour désigner les algorithmes qui maintiennent cette propriété à chaque étape et "nearsharp" pour les autres. La différence avec la méthode du simplexe provient du fait qu'à chaque itération les variables duales sont mises à jour immédiatement par la

---

[1] Dans ce sens, la liste Q joue le rôle du critère de sélection de la variable d'entrée, en anglais "pricing step".

visite de la sous-arborescence dont j est la racine tandis qu'ici cette correction est reportée à plus tard avec la liste Q. Le choix de la structure de données pour la liste Q est crucial car il détermine la complexité de l'algorithme. À cet égard, il convient de rappeler que la politique d'insertion-suppression des sommets dans Q dirige la fouille du graphe pour atteindre les conditions d'optimalité. Le reste de ce chapitre est consacré à l'étude des différentes stratégies d'exploration du graphe pour la recherche des chemins les plus courts.

Afin de mesurer la complexité théorique du temps d'exécution des algorithmes, nous adoptons la mesure de la complexité dans le pire des cas sur une machine à accès direct (Tarjan, 1983). Nous calculons le nombre maximum d'instructions élémentaires, sur tous les ensembles de données possibles de taille n et m pour un algorithme donné. Pour ce faire, nous posons q_0, q_1 et q_2 tels que q_0 représente la complexité de l'initialisation de l'algorithme, q_1 est le temps requis à retrancher un élément de la liste Q et, q_2 est le temps nécessaire pris pour l'insertion d'un sommet dans Q (voir Gallo-Pallottino, 1988). De plus, si c_1 est le nombre de fois qu'un sommet est supprimé de Q et si c_2 est le nombre de fois qu'un sommet est ajouté à Q, alors la complexité d'un algorithme de recherche des chemins minimaux est $O(q_0 + c_1q_1 + c_2q_2)$.[1] D'une manière générale, si T et f sont des fonctions de x, alors "$T(x) = O(f(x))$" signifie qu'il existe une constante positive k et une valeur $\bar{x}$ telle que $T(x) < kf(x)$ pour tout $x > \bar{x}$. En d'autres mots, si x est une mesure de la taille d'un problème, la croissance asymptotique du temps de calcul T(x) est de l'ordre de f(x) au plus[2]. Nous déduisons une bonne borne supérieure pour les valeurs c_1 et c_2 en posant $\tilde{c}$ comme étant le nombre maximum de fois qu'un sommet est amélio-

---

[1] Lire "grand O de $q_0 + c_1q_1 + c_2q_2$" ou plus simplement "ordre de $q_0 + c_1q_1 + c_2q_2$".

[2] Réciproquement, on dit que $T(x) = \Omega(f(x))$ s'il existe une constante positive k et une valeur $\bar{\bar{x}}$ telle que $T(x) > kf(x)$ pour tout $x > \bar{\bar{x}}$.

ré. Puisque $\tilde{c}$ est plus grand ou égal à un, nous avons $n \leq c_1 \leq n\tilde{c}$ et $m \leq c_2 \leq$ Min $\{nc_1, m\tilde{c}\}$. Nous soulignons que l'initialisation de l'algorithme requiert toujours au moins O(n) affectations. C'est dire que q_0 est $\Omega(n)$.

Cette façon d'unifier les algorithmes des chemins les plus courts se différencie de la classification traditionnelle "label setting", "label correcting" employée par plusieurs auteurs, entre autres, Gilsinn et Witzgall (1973), Dial et al. (1979) et Glover et al. (1985b). Dans cette classification, les algorithmes dont le choix à chaque itération est de prendre le sommet de Q qui possède l'étiquette la plus faible sont des algorithmes par extension sélective et les autres sont des algorithmes par ajustement progressif.

Les algorithmes par extension sélective ("label setting") s'appliquent seulement aux graphes munis de longueurs non négatives et bénéficient de l'avantage qu'une fois retiré de la liste Q, un sommet n'y revient plus. Cela nous permet de conclure qu'un algorithme par extension sélective termine en exactement n itérations et, au pire des cas, il requiert $O(m+n^2)$ opérations élémentaires. À chaque itération un sommet reçoit une étiquette permanente qui représente la longueur minimale pour s'y rendre. La caractéristique de ces algorithmes est qu'ils nécessitent peu d'itérations, mais le travail à chaque itération est coûteux. Les algorithmes de cette classe se distinguent entre eux par la manière dont s'opère la recherche de l'élément minimum dans la liste Q.

D'autre part, les algorithmes par ajustement progressif ("label correcting") emploient des règles d'insertion-suppression des sommets dans la liste Q très rapides à exécuter, par exemple, premier entré, premier sorti. Par contre, il n'y a aucune raison théorique qui limite à 1 le nombre de fois qu'un sommet est retiré de la liste Q. Il existe des algorithmes pour lesquels le nombre théorique d'insertions est $O(2^n)$. Contrairement aux algorithmes par extension sélective, les algorithmes par ajustement progressif itèrent un grand nombre de fois, mais le travail à chaque itération est peu coûteux. Ils s'appliquent aussi aux graphes munis de

longueurs négatives. Pour ces algorithmes, la longueur minimale d'un chemin allant de r à i, pour tout i, est connue seulement quand la liste Q est vide.

3.8 Recherche du minimum d'abord

Le problème du chemin le plus court a été résolu de façon analogue par de nombreux auteurs. Le plus rapide moyen de calcul semble être l'algorithme proposé par Dijkstra (1959). Pollack et Wiebenson (1959) attribuent à Minty la méthode que nous présentons. Il consiste à sélectionner le sommet de Q qui possède l'étiquette de valeur minimum. Lorsque les longueurs des arcs sont non négatives, l'algorithme procède en exactement n itérations. Au début de chacune des itérations, l'ensemble des sommets est partitionné en deux sous-ensembles disjoints, Q et $\bar{Q} = N-Q$. Initialement, $Q = N-\{r\}$ et $\bar{Q} = \{r\}$. Chaque sommet i de N est affecté d'une étiquette Π_i qui vérifie la propriété suivante :

Propriété 1

$$\Pi_i = \begin{cases} \Pi_i^* & \text{si } i \in \bar{Q} \\ \underset{k \in \bar{Q} \cap \Gamma_i^{-1}}{\text{Min}} \{\Pi_k + \ell_{ki}\} & \text{si } i \in Q \end{cases}$$

En d'autres mots, la valeur Π_i, $i \in Q$ donne la valeur minimale des chemins issus du sommer r jusqu'au sommet i, soumis à la condition que tous ses sommets sont dans $\bar{Q}$, sauf i. La pertinence de l'algorithme dépend alors du lemme suivant :

Lemme 3.1

Soit j dans Q tel que $\Pi_j = \underset{i \in Q}{\text{Min}} \{\Pi_i\}$

alors $\Pi_j = \Pi_j^*$

Preuve Puisque nous supposons le réseau R fortement connexe, il existe un chemin du sommet r au sommet j de longueur Π_j. Pour montrer que ce

chemin est le chemin le plus court entre les sommets r et j, considérons un chemin μ allant de r à j :

$$\mu = r, i_1, i_2, \ldots, h, i_k, \ldots, j$$

Divisons-le en deux parties : μ_1 la partie commençante du chemin allant de 1 à h où h est le premier sommet de Q rencontré, et μ_2 le reste. Par hypothèse, nous avons les inégalités suivantes :

$$d(\mu_1) > \Pi_h > \Pi_j$$

Finalement, comme nous supposons la longueur des arcs non négatives, la longueur du chemin μ_2 est plus grande ou égale à zéro. Alors, nous déduisons que la longueur du chemin μ est plus grande ou égale à Π_j.

■

L'algorithme détermine l'un après l'autre les sommets les plus proches du sommet r. À cause de cette propriété, l'algorithme de Dijkstra est aussi connu sous le nom anglais "label setting method" ou méthode par extension sélective en français. Aussi, on dit que les sommets dans $\bar{Q}$ sont des sommets permanents. Conséquemment, si l'on n'est intéressé qu'au chemin le plus court du sommet r à un autre sommet donné t, l'exécution de l'algorithme est interrompue quand le sommet t est atteint.

L'algorithme de Dijkstra prend avantage que les longueurs des arcs sont non négatives. Dans le cas où cette condition n'est pas remplie, la complexité de la méthode est $O(n2^n)$ (Johnson, 1973). Pour cette raison, nous poursuivons notre discussion en supposant que les arcs sont munis de longueurs plus grandes ou égales à zéro. Dans ce cas, quelques-unes des implantations de la méthode s'appuient sur les deux propriétés suivantes :

Propriété 2

Posons $lmax = \underset{(i,j)\in U}{Max} \{\ell_{ij}\}$. Si, à une itération donnée, j représente le sommet dont l'étiquette est la plus petite parmi les sommets de l'ensemble Q, alors

- $\Pi_j \geq \Pi_i$ pour tout i dans $\bar{Q}$,
- $\Pi_j \leq \Pi_k \leq \Pi_j + lmax$ pour tout k dans Q et Π_k est une valeur finie.

Preuve Puisque la valeur Π_k, $k\in Q$, représente la longueur d'un chemin minimal allant de r jusqu'au sommet k, soumis à la condition que tous ses sommets sont dans $\bar{Q}$ sauf k, donc la valeur Π_k vaut la marque d'un sommet permanent plus la longueur d'un arc.

∎

Propriété 3

Posons $m_k = \underset{i\in Q}{Min} \{\Pi_i\}$ à l'itération k. Alors la suite $m_1, m_2, \ldots, m_n$ est une suite monotone non décroissante.

Preuve La preuve découle directement de la propriété 2.

∎

Puisque le graphe est représenté par la liste des successeurs, chaque arc est analysé exactement une fois car chaque sommet i est retiré une seule fois de la liste Q. C'est dire que c_1 vaut n et c_2 vaut m. Avec cette représentation du graphe en machine, le temps pris par l'algorithme de Dijkstra est $O(q_0 + nq_1 + mq_2)$.

La complexité de cette méthode dépend alors de la structure de données employée afin d'obtenir le sommet d'étiquette minimum. Une file de priorité ("priority queue") est une structure de données abstraite utilisée

pour représenter un ensemble d'éléments, chaque élément étant déterminé par un identificateur et une valeur numérique réelle. La valeur numérique associée à un élément est la clé de l'élément ou l'étiquette. Les opérations possibles sur cette structure de données sont principalement (Tarjan, 1983; Ahuja et al, 1988) :

- INSERT (i,Q) Insertion d'un nouvel élément i dans la file de priorité Q.
- DELETEMIN (Q) Identification et suppression de l'élément ayant la valeur numérique la plus faible. La procédure retourne cet élément.
- INIT (Q,S) Initialisation d'une file de priorité avec les éléments de l'ensemble S.
- REINSERT (j,Q) Remise à jour de la file de priorité lorsque la valeur de la clé du sommet j est modifiée.

La définition d'une file de priorité nous permet de préciser les opérations "$Q \leftarrow Q - \{i\}$" et "$Q \leftarrow Q \cup \{j\}$" énoncés précédemment. Dans ces conditions, la méthode de Dijkstra prend la forme suivante :

ALGORITHME 4 (Dijkstra)

```
INIT (Q,{r})
faire
  début
    i ← DELETEMIN (Q)
    Pour chaque (i,j)∈Λ_i faire
          si Π_j > Π_i + ℓ_ij alors
             début
               Π_j ← Π_i + ℓ_ij
               p_j ← i
               si j∉Q alors INSERT (j,Q)
               sinon REINSERT (j,Q)
             fin
  fin
tant que Q≠∅.
```

3.8.1 Algorithme SDKSTR

Initialement, il a été suggéré de représenter la file de priorité par une liste. Une liste peut être triée ou non triée. La recherche de l'élément minimum d'une liste triée est triviale : il suffit de prendre le premier élément de la liste. En revanche, les insertions et les réinsertions nécessitent un parcours de la moitié de la liste en moyenne, pour conserver l'ordre dans celle-ci. D'un autre côté, la liste non triée facilite l'insertion de nouveaux éléments, mais rend plus complexe la recherche des éléments minimaux. Considérons une liste non triée. La recherche du sommet minimum consiste à parcourir tous ses éléments. À la $k^{ième}$ itération, la liste contient au plus n-k sommets. Pour tout l'algorithme, le nombre total de comparaisons requises pour l'opération DELETEMIN est

$$\sum_{k=1}^{n-1} (n-k) = (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}$$

En outre, les opérations d'initialisation et d'insertion requièrent O(1) calcul. Remarquons qu'il n'est pas nécessaire de remettre à jour la file de priorité lorsque la valeur de la clé d'un sommet est modifiée car on utilise une liste non triée. Donc, le nombre de réinsertions est zéro. Donc, nous concluons que la complexité de l'algorithme de Dijkstra est $O(m) + O(n^2)$ quand la file de priorité est une liste non triée. Par la suite, nous désignons par SDKSTR l'algorithme de Dijkstra usant d'une liste non triée.

Dans un graphe dense, où m est de l'ordre de n^2, le second terme est tolérable, par contre lorsque le graphe est peu dense, $m << n^2$, il est préférable d'utiliser une autre structure de données.

3.8.2 Arbres partiellement ordonnés, algorithme SHEAP

Que nous utilisions des listes triées ou non, le temps nécessaire à la réalisation de l'une quelconque des deux opérations d'ajout ou de retranchement sur un ensemble de k éléments est proportionnel à k. Or, il existe une représentation des données dans laquelle ces opérations ont une

complexité de $O(\log_d k)$. Le principe consiste à placer les éléments de la file de priorité dans un arbre d-aire complet partiellement ordonné.

Un arbre partiellement ordonné (voir figure 3.3) est un arbre tel que la clé d'un noeud est toujours plus petite ou égale aux clés de ses successeurs. Par la règle de transitivité, l'élément dont la clé est la plus faible se trouve à la racine de l'arbre.

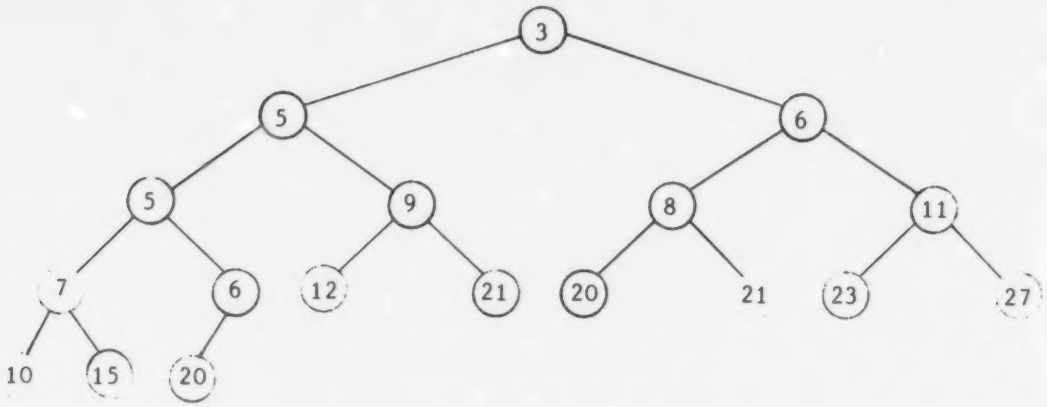


Figure 3.3 : Un arbre partiellement ordonné. Les nombres dans les noeuds sont les clés.

Un arbre d-aire est un arbre dont chaque noeud possède au plus d successeurs. Une feuille est un noeud qui n'a pas de successeurs. Un arbre d-aire complet est un arbre d-aire dont les seules feuilles manquantes se trouvent à droite des feuilles situées au plus bas niveau de l'arbre. La figure 3.4 montre un arbre binaire complet et partiellement ordonné. Gondran et Minoux (1985) emploient le terme "tas" pour désigner cette structure de données quand d vaut deux.

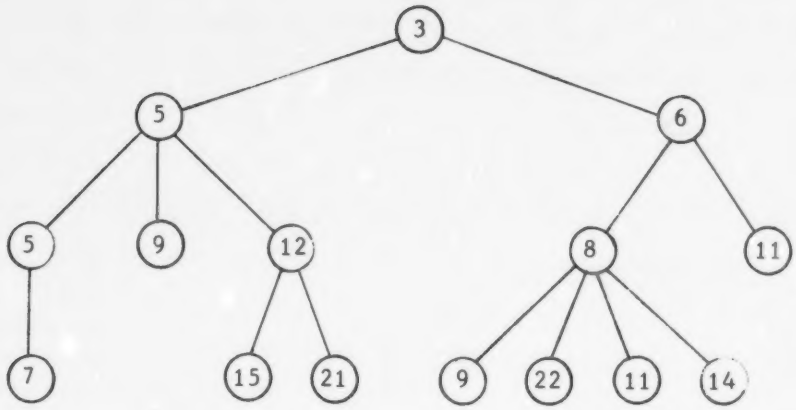


Figure 3.4 : Un tas

Avec cette organisation des données dans la file de priorité, l'insertion d'un nouvel élément et la diminution de la clé d'un élément requièrent $O(\log_d k)$ tandis qu'ôter l'élément ayant la clé la plus faible nécessite $O(d\log_d k)$ comparaisons où k est le nombre d'objets dans la file de priorité. Ceci vient du fait qu'aucun chemin allant de la racine à un noeud quelconque n'a une longueur supérieure à $1 + \lfloor \log_d k \rfloor$. Cette structure de données a été abondamment étudiée dans la littérature. Nous référons le lecteur qui désire connaître les détails liés à la manipulation de ce type de file de priorité à Tarjan (1983).

Nous vérifions aisément que la complexité de l'algorithme de Dijkstra est $O((m+nd)\log_d n)$ si la file de priorité est un arbre d-aire complet partiellement ordonné. Tarjan (1983) souligne que l'opération DELETEMIN est d fois plus coûteuse que les opérations INSERT et REINSERT, alors un choix judicieux de d est $\text{Max}(2, \lceil \frac{m}{n} \rceil)$.

La motivation de ce choix repose sur la remarque suivante. Quelque soit la densité du graphe, le nombre d'appels à DELETEMIN est n tandis que plus le graphe est dense, plus de fois l'algorithme fait appel à REINSERT. Or, cette dernière procédure est d fois moins coûteuse que DELETEMIN.

Ainsi, il est possible de gagner du temps en augmentant la valeur de d. Par contre, des expérimentations numériques menées par Johnson (1972) ont révélé que la valeur de d qui minimise le nombre de comparaisons est deux. Par conséquent, nos expérimentations numériques ont été conduites avec l'algorithme SHEAP avec d valant deux. Pour de plus amples détails, se référer au chapitre suivant.

3.8.3 Adressage dispersé - Algorithme SDIAL

Le temps requis pour positionner correctement un sommet dans la structure de données précédente est assez considérable. Par conséquent, plusieurs auteurs ont concentré leurs efforts à l'élaboration d'une file de priorité qui permet cette opération en temps constant. Dans le cadre de la recherche des chemins minimaux avec la méthode de Dijkstra, une fonction d'adressage dispersé ("hashing") est une fonction h() entière non décroissante qui projette l'ensemble des étiquettes des sommets sur l'ensemble des entiers 0,1,2,...,K où K est une constante plus grande ou égale à zéro. Le compartiment c entrepose tous les sommets i dans la file de priorité tel que $h(\Pi_i) = c$. La figure 3.5 illustre l'organisation des données dans un schéma d'adressage dispersé.

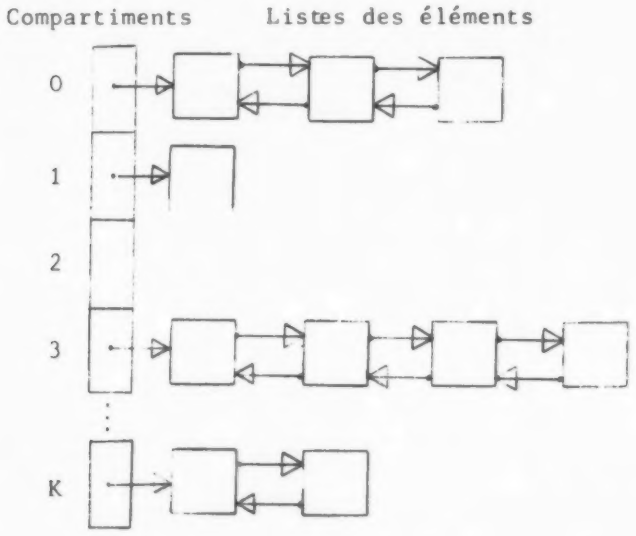


Figure 3.5 : Organisation des données dans un schéma d'adressage dispersé

À l'intérieur d'un compartiment, les éléments sont doublement chaînées pour faciliter leur transfert d'un compartiment à un autre.

D'une manière générale, le sommet dont l'étiquette est la plus faible est déterminée en examinant séquentiellement les compartiments jusqu'au premier compartiment non vide. Ensuite, ce dernier est parcouru afin d'y trouver le sommet cherché. Un nouvel élément est inséré dans la file de priorité en l'ajoutant dans le compartiment approprié. De même, lorsque l'étiquette d'un sommet est améliorée, ce dernier est transféré dans le bon compartiment, suivant la valeur de sa nouvelle marque. La fouille des compartiments débute avec le compartiment 0 et, à chaque étape, elle reprend là où elle avait laissée. L'algorithme termine quand tous les compartiments sont épuisés.

Denardo et Fox (1979) proposent de stocker les sommets dans des compartiments dont la taille est w. Le $c^{\text{ième}}$ compartiment contient tous les sommets dont les étiquettes appartiennent à l'intervalle $[w \cdot c, w \cdot (c+1) - 1]$ (la numérotation des compartiments commence à zéro). Le nombre de compartiments étant crucial, la pertinence de cette implantation repose sur l'observation suivante. À cause de la propriété 3, lorsque la fouille des compartiments s'arrête à un compartiment c, les compartiments 0,1,2,...,c-1 sont vides et ne sont plus jamais utilisés. Alors la propriété 2 nous invite à considérer la fonction d'adressage modifiée

$$h' = h \text{ Mod } (lmax + 1) .$$

Dans ces conditions, à l'itération k, le compartiment c contient les sommets dont les étiquettes appartiennent à l'intervalle

$$[\, lmax \cdot k + w \cdot c,\ lmax \cdot k + w \cdot (c+1) - 1]^{-1}$$

Alors, le nombre de compartiments nécessaires est $\left\lceil \frac{lmax + 1}{w} \right\rceil$.

Un sommet i est inséré dans la file de priorité en l'ajoutant dans le compartiment approprié, ce dernier étant obtenu en O(1) calcul par la formule

$$c = \left\lceil \frac{\Pi_i \text{ Mod } (\text{lmax} + 1)}{w} \right\rceil .$$

La fonction d'adressage h' prend avantage que la procédure DELETEMIN retourne une suite non décroissante de valeurs, pour réduire l'espace mémoire requis. Ces auteurs ont aussi montré que si w vaut lmin, où lmin = $\underset{(i,j)\in U}{\text{Min}} \{\ell_{ij}\}$, alors tous les sommets i du premier compartiment sont tels que Π_i vaut Π_i^*. Malheureusement, cette technique ne fonctionne pas si la valeur de lmin est zéro.

D'après plusieurs expérimentations numériques (Dial et al., 1979), l'implantation la plus efficace de la méthode de Dijkstra est due à Dial (1969). Cet algorithme que nous nommons SDIAL, utilise lmax + 1 compartiments de taille 1 et nous en proposons une mise en oeuvre au chapitre suivant. Toutefois, il convient de souligner que cet algorithme s'applique uniquement aux graphes munis de longueurs entières (non négatives). Dans le cas où cette condition n'est pas remplie, Fox (1978) suggère une conversion des données en multipliant les nombres par une grande constante en vue d'éliminer la partie fractionnaire. Ainsi, dans l'algorithme SDIAL, le premier compartiment non vide contient tous les sommets dont la marque est la plus faible parmi tous les sommets de la file de priorité. L'initialisation des lmax + 1 compartiments requiert O(lmax) affectations tandis que les opérations INSERT et REINSERT se font en O(1) calcul. L'opération la plus coûteuse est DELETEMIN qui, au pire cas, nécessite le balayage de lmax compartiments. Par conséquent, la complexité de l'algorithme SDIAL est O(m+n lmax).

Il est intéressant d'ajouter à cette discussion qu'un cas encore plus particulier est celui où toutes les longueurs des arcs valent un. C'est le cas, pourtant assez fréquent, de la recherche d'un chemin le plus court en nombre d'arcs dans un graphe. D'après les résultats ci-haut, ce calcul peut se faire en O(m+n).

3.8.4 Algorithme SRHEAP

Très récemment, Ahuja et al. (1988) ont proposé une nouvelle méthodologie quant à l'utilisation des compartiments à l'intérieur de l'algorithme de Dijkstra lorsque les longueurs des arcs sont des nombres entiers (non négatifs). Notre exposé diffère quelque peu du texte original fourni par ces auteurs. En premier lieu, nous présentons notre interprétation de la méthode. Ensuite, nous décrivons l'implantation suggérée par Ahuja et al. (1988) et, au chapitre suivant, nous montrons une mise en oeuvre informatique efficace.

Conceptuellement, l'algorithme consiste à maintenir les sommets de la file de priorité dans $2 \cdot lmax + 1$ compartiments de taille 1, rangés dans un arbre, appelons-le RHEAP (pour "redistributive heap") de $2 + \lceil \log lmax \rceil$ niveaux. Dans cet arbre, les noeuds sont composés de deux cellules. La cellule gauche contient un numéro correspondant à l'ordre de visite de l'arbre et l'autre un pointeur vers la liste des sommets associés à ce noeud. L'arbre est exploré de haut en bas, niveau par niveau, de gauche à droite. La figure 3.6 illustre l'arbre en question.

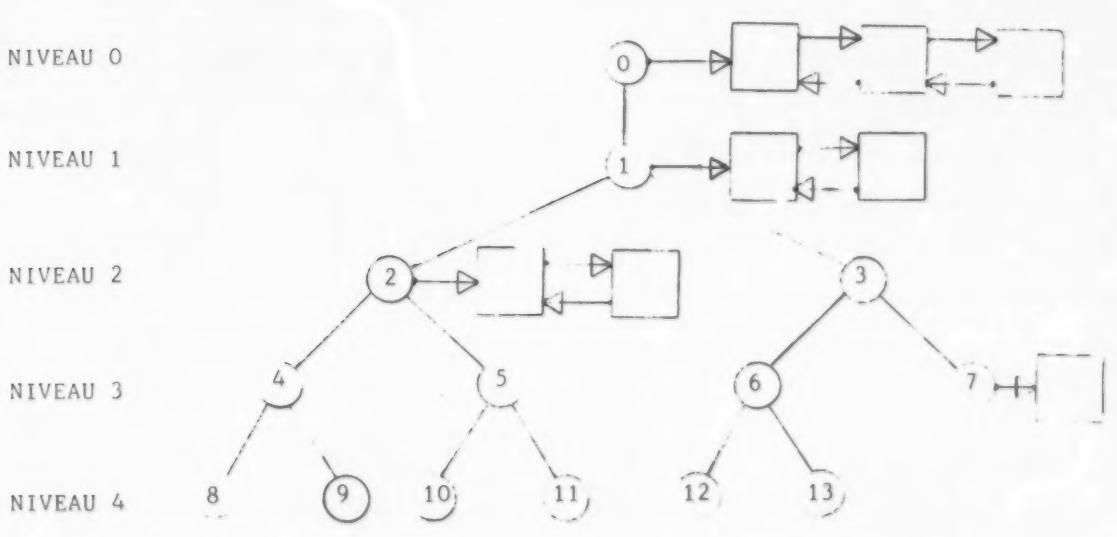


Figure 3.6 : Arbre Rheap (lmax = 6)

Par définition d'un arbre RHEAP, le nombre de noeuds d'un niveau n_c est plus petit ou égal au nombre de noeuds des niveaux $0,1,2,\ldots,n_c-1$. Contrairement à la méthode de Dial où les compartiments sont balayés d'une manière circulaire avec la fonction d'adressage dispersé $h' = \Pi_i$ Mod(lmax+1), nous attachons au noeud c de l'arbre tous les sommets de la file de priorité selon la règle

$$c = \begin{cases} \Pi_i - \text{base} & \text{si } \Pi_i < 2^{\lceil \log \text{base} \rceil} \\ \Pi_i & \text{sinon .} \end{cases}$$

Le logarithme est pris en base 2.

Nous précisons le rôle de la variable **base** dans la procédure DELETEMIN suivante :

Procédure DELETEMIN

1 Recherche du premier compartiment non vide.
Poser c=0
Tant que le compartiment c est vide faire c ← c+1.

2 Si c=0 ou si c=1 alors retirer un sommet de ce compartiment.STOP.
Sinon aller à 3.

3 Poser base ← base + c
Réinsérer les sommets dans les compartiments selon la règle énoncée ci-haut. Aller à 1.

Initialement, **base** vaut zéro (log0 = infini).

Avec cette structure de données, l'insertion et la réinsertion des sommets dans la file de priorité sont triviales tandis que la procédure DELETEMIN consiste à redistribuer les sommets de la file de priorité afin d'aspirer le ou les sommets d'étiquettes minimales dans le compartiment 0. De plus, les propriétés 2 et 3 de la méthode de Dijkstra nous assurent que 2.lmax+1 compartiments sont nécessaires et suffisants pour emmagasiner les sommets de la file de priorité lorsque la fonction d'adressage est la règle énoncée ci-haut.

Du point de vue pratique, Ahuja et al. (1988) ont proposé quelques stratégies d'implantation qui accélèrent la procédure DELETEMIN. La plus simple consiste à utiliser K+1, où $K=1+\lceil \log lmax \rceil$, gros compartiments qui correspondent aux niveaux de l'arbre Rheap. Afin de clarifier cette structure de données, nous définissons $[l^k,u^k]$ comme étant l'intervalle d'admission des sommets dans le compartiment k. Alors nous avons

$$b^k = \{i \in Q \text{ tel que } \Pi_j \in [l^k,u^k]\}.$$

Les sommets ne sont pas triés à l'intérieur d'un compartiment, par contre ils sont liés par une liste doublement chaînée. De plus, nous employons une variable $assign_i$ qui indique le compartiment où se trouve le sommet i.

L'algorithme débute avec

$$[l^k,u^k] = \begin{cases} [0,0] & \text{si } k=0 \\ [1,1] & \text{si } k=1 \\ [2^{k-1},2^k-1] & \text{si } k=2,3,\ldots,K \end{cases}$$

$$b^k = \begin{cases} \{r\} & \text{si } k=0 \\ \emptyset & \text{sinon.} \end{cases}$$

La procédure DELETEMIN consiste à prendre soit un sommet dans le compartiment b^0, ou si ce dernier est vide dans le compartiment b^1. Autrement, si ceux-ci sont vides, nous cherchons le premier compartiment non vide d'indice minimum. Soit pntr cet indice et dmin la valeur de l'étiquette la plus faible dans le compartiment b^{pntr}. Après avoir redéfini les intervalles des compartiments $b^0,b^1,\ldots,b^{pntr-1}$ par la formule suivante :

$[l^0,u^0] \leftarrow [dmin,dmin]$

$[l^1,u^1] \leftarrow [dmin+1,dmin+1]$

si pntr=K alors $[l^k,u^k] \leftarrow [2^{k-1}+dmin,2^k-1+dmin]$ pour k=2,3,...,K

sinon $[l^k,u^k] \leftarrow [2^{k-1}+dmin,\min\{2^k-1+dmin,u^{pntr}\}]$ pour k=2,3,...,pntr

Nous redistribuons les sommets du compartiment pntr dans ceux-ci. Notons que si pntr est plus petit que K alors l^{pntr} est plus grand que u^{pntr}. Dans ce cas le compartiment pntr est inactif. Quand la marque d'un sommet est améliorée, la nouvelle valeur est toujours plus petite que l'ancienne Une procédure possible pour réinsérer les sommets est la suivante :

Procédure REINSERT (j)

pntr ← $assgn_j$

$b^{pntr} \leftarrow b^{pntr} - \{j\}$

tant que $\Pi_j \notin [l^{pntr}, u^{pntr}]$ faire pntr ← pntr - 1

$b^{pntr} \leftarrow b^{pntr} \cup \{j\}$

$assgn_j$ ← pntr.

D'une façon identique, nous insérons un sommet pour la première fois dans la file de priorité en faisant :

Procédure INSERT (j)

pntr ← K

tant que $\Pi_j \notin [l^{pntr}, u^{pntr}]$ faire pntr ← pntr - 1

$b^{pntr} \leftarrow b^{pntr} \cup \{j\}$

$assgn_j$ ← pntr.

La procédure termine lorsque tous les compartiments sont libres.

Le bon fonctionnement de l'algorithme repose sur quelques propriétés satisfaites par la structure de données. De toute évidence, nous avons :

- Si le sommet j est présent dans la file de priorité, alors $\Pi_j \in [l^0, u^K]$.
- Si $k_1 < k_2$ et si les compartiments k_1 et k_2 sont actifs, alors $u^{k_1} < l^{k_2}$. En d'autres termes, si $i_1 \in b^{k_1}$ et si $i_2 \in b^{k_2}$ alors $\Pi_{i_1} < \Pi_{i_2}$.

- L'intersection des intervalles des compartiments actifs est vide. D'autre part, l'union de ces intervalles est l'intervalle fermé $[l^0,u^K]$.

Par la suite, nous donnons à cette implantation particulière de l'algorithme de Dijkstra le non SRHEAP.

La complexité théorique de l'algorithm SRHEAP est O(m+nloglmax). Pour le vérifier, résumons les étapes effectuées par l'algorithme.

L'étape **initialisation** est composée du calcul de la valeur lmax et de la création des compartiments. La première requiert O(m) comparaisons et il y a O(loglmax) compartiments à initialiser. Donc, la complexité de cette étape est O(m+loglmax).

Pour un sommet j quelconque, j≠r, le temps total pris par les procédures **INSERT** et **REINSERT** est O(loglmax). En effet, si ce sommet est inséré pour la première fois dans la file de priorité dans le compartiment k_1, cela nécessite $O(K-k_1)$ comparaisons. Ensuite, ce sommet est réinséré (l'étiquette de ce sommet est diminuée) tant qu'il ne parvient pas au compartiment 0. Cela requiert $O(k_1)$ comparaisons. Puisque la liste des sommets appartenant à un compartiment est doublement chaînée, nous retirons un sommet d'un compartiment en O(1) calcul. Donc, pour tout l'algorithme, le temps pris à insérer et à réinsérer les sommets est O(nloglmax).

La procédure **DELETEMIN** est exécutée n fois. Si l'un ou l'autre des compartiments b^0 ou b^1 est non vide, le minimum est obtenu en O(1) effort. Autrement, la recherche du premier compartiment non vide est bornée par O(loglmax). Pour tout l'algorithme cela représente O(nloglmax). Soit pntr, l'indice du premier compartiment non vide. Les sommets de ce compartiment sont balayés afin d'y trouver la valeur dmin. Il y au plus O(n) sommets à comparer. Or, les sommets du compartiment pntr sont réinsérés dans les compartiments dont les indices sont inférieurs à pntr. Au

pire des cas, cela se produit O(loglmax). Donc, pour tout l'algorithme, le calcul de la valeur dmin est O(nloglmax).

Finalement, chaque arc est analysé exactement une fois, par conséquent, cela ajoute O(m) comparaisons et la complexité totale de l'algorithme est O(2m+4nloglmax).

3.8.5 Transformation des longueurs

Lorsque nous ne sommes intéressés qu'au chemin le plus court du sommet r à un autre sommet donné t, l'exécution de l'algorithme 4 est interrompue quand le sommet t est atteint. Dans cette sous-section, nous montrons comment accélérer la procédure de Dijkstra dans le cas où le chemin le plus court de r à t, et seulement lui, nous intéresse et que l'on dispose d'une borne inférieure de la longueur du chemin le plus court entre tous les sommets i, $i \in N$ et t. Dans un réseau routier, par exemple, la distance euclidienne fournit cette borne. Le principe de cette heuristique repose sur une transformation des longueurs qui conserve les chemins les plus courts.

Proposition 3.2

Si T_r^* est une arborescence des chemins les plus courts lorsque le coût des arcs est $\ell_{ij}, (i,j) \in U$, alors T_r^* demeure optimale lorsque le coût des arcs est remplacé par $\ell'_{ij} = \ell_{ij} - b_{it} + b_{jt}$ où b_{it} est une borne inférieure pour $d(h^*_{it})$, telle que $0 \leq b_{it} \leq \ell_{ij} + b_{jt}$.

Preuve Soit $h_{rs} = (r = s_0, s_1, \ldots, s_{k-1}, s_k = s)$ un chemin allant de r à s. La longueur totale de ce chemin avec les longueurs transformées est

$$d'(h_{rs}) = \sum_{(i,j)\in h_{rs}} \ell'_{ij} = \sum_{(i,j)\in h_{rs}} \ell_{ij} - b_{it} + b_{jt}$$

$$= \sum_{(i,j)\in h_{rs}} \ell_{ij} - b_{s_0 t} + b_{s_1 t} - b_{s_1 t} + \ldots - b_{s_{k-1} t} + b_{st}$$

$$= d(h_{rs}) - b_{rt} + b_{st}.$$

Donc, pour tout chemin allant de r à s, l'ancienne et la nouvelle longueur diffèrent d'une constante indépendante du chemin suivi.

∎

Un corollaire immédiat est $\Pi'^{*}_j = \Pi^{*}_j - b_{rt} + b_{jt}$, où Π'^{*}_j représente la plus courte distance entre les sommets r et t avec les longueurs ℓ'_{ij}. Aussi, nous remarquons que ℓ'_{ij} est plus grand ou égale à zéro pour tous les arcs du réseau. Dans ces conditions, nous pouvons appliquer la méthode de Dijkstra en substituant aux longueurs originales du réseau les longueurs ℓ'_{ij}. Ainsi, à chaque itération, nous cherchons le sommet k qui minimise la somme $\Pi_i + b_{it}$. L'algorithme termine quand le sommet t est atteint. En mettant en œuvre cette modification de l'algorithme de Dijkstra nous espérons atteindre le sommet t plus rapidement. L'utilisation des longueurs transformées revient à favoriser une direction de recherche. Bovet (1986) soutient que la région explorée dans la méthode modifiée est assimilable à une figure ressemblant à une larme dont la pointe se trouve en t. D'après cet auteur, cette extension, simple à implanter, permet des gains de calcul pouvant aller jsuqu'à 35 % par rapport au temps de calcul de la méthode originale. D'autre part, Sedgewick et Vitter (1984) ont montré que la complexité théorique de la méthode modifiée est O(n) pour certains modèles de graphe. Or d'une manière générale, la complexité théorique n'est pas améliorée, c'est pourquoi nous qualifions cette procédure d'heuristique.

3.9 Recherche en largeur d'abord - Algorithme LQUEUE

En toute généralité, lorsque les longueurs des arcs sont des nombres quelconques, la méthode de Dijkstra ne fonctionne pas. Nous créditons à Ford (1959), Moore (1956) et Bellman (1958) (voir Dreyfus, 1969) la première procédure capable de résoudre ce problème plus général.

L'algorithme en question est une méthode itérative qui consiste à donner des étiquettes aux sommets. À la fin de l'itération k, ces étiquettes correspondent aux longueurs des chemins les plus courts formés de k

arcs au plus. Le symbole π_j^k désigne la longueur du chemin allant de r à j après k itérations. La méthode d'étiquetage consiste à poser

$$\pi_j^{k+1} = \text{Min}\ \{\pi_j^k,\ \underset{i\in\Gamma_j^{-1}}{\text{Min}}\{\pi_i^k + \ell_{ij}\}\}.$$

Sous l'hypothèse d'absence de circuit absorbant, nous sommes certains qu'au plus n-1 itérations sont nécessaires pour avoir

$$\pi_j^{n-1} = \pi_j^* \quad \text{pour tout } j\in N.$$

Il convient de souligner que la condition nécessaire pour que π_j^{k+1} soit meilleure que π_j^k est qu'il existe au moins un sommet $i\in\Gamma_j^{-1}$ tel que

$$\pi_i^k < \pi_i^{k-1} .$$

En effet, si tous les prédécesseurs de j n'ont pas reçu d'amélioration à l'itération k, nous avons

$$\pi_j^{k+1} = \pi_j^k .$$

Nous améliorons la procédure de marquage en cherchant les arcs améliorants issus des prédécesseurs de j qui ont reçu une amélioration à l'étape précédente. Pour ce faire, nous symbolisons par Q^k l'ensemble des sommets améliorés à l'itération k :

$$Q^k = \{i \mid \pi_i^k < \pi_i^{k-1}\} .$$

Alors, la nouvelle procédure de marquage est :

$$\pi_j^{k+1} = \text{Min}\ \{\pi_j^k,\ \underset{i\in\Gamma_j^{-1}\cap Q^k}{\text{Min}}\{\pi_i^k+\ell_{ij}\}\} = \underset{i\in\Gamma_j^{-1}\cap Q^k}{\text{Min}}\{\pi_j^k, \pi_i^k+\ell_{ij}\}$$

D'autre part, pour un sommet $i\in Q^k$ donné, la k+i$^{\text{ième}}$ opération de marquage devient :

$$(1) \qquad \pi_j^{k+1} = \underset{j\in\Gamma_i}{\text{Min}}\ \{\pi_j^k, \pi_i^k+\ell_{ij}\}.$$

Les sommets j améliorés durant l'itération k+1 sont placés dans Q^{k+1}. En outre, la façon dont les sommets sont insérés ou retirés de l'ensemble Q^k n'influence pas le bon fonctionnement de l'algorithme. Ceci provient du fait que l'analyse des sommets de Q^{k+1} ne débute pas tant que Q^k est non vide.

Une file est une liste dans laquelle le premier élément entré est le premier à sortir. En anglais, cette structure de données est bien connue sous l'acronyme FIFO, qui signifie "first in, first out". Or, nous vérifions aisément que la gestion des ensembles $Q, Q^1, \ldots, Q^k, \ldots$ est équivalent à l'utilisation d'une file à la fin de laquelle nous insérons les sommets j améliorés par la procédure de marquage 1 (voir page précédente). Le choix à chaque étape étant de prendre le premier élément de la liste. En effet, il suffit de constater qu'après avoir retiré du début de la file tous les sommets $i \in Q^k$, la file est formée de tous les j de l'ensemble Q^{k+1}. Par la suite, nous référons à LQUEUE pour désigner l'algorithme obtenu lorsque Q est une file.

Puisque Π_i^k représente la longueur du chemin le plus court allant de r à i formé de k arcs au plus et que le réseau ne recèle pas de circuit absorbant, l'étiquette d'au moins un sommet prend sa valeur optimale à chaque étape. Donc, au pire des cas, il y a

$$1 + (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2} + 1$$

insertions dans la liste L. C'est dire que le plus grand nombre de fois qu'un sommet est choisi est n-1. Donc $\tilde{c} = n-1$, alors c_1 est $O(n^2)$ et c_2 est $O(\text{Min}\{n^3, mn\})$. Il est bien connu que l'insertion et la suppression des sommets dans une liste sont triviales, par conséquent, la complexité algorithmique de l'algorithme LQUEUE est $O(mn)$ si le graphe est peu dense et $O(n^3)$ sinon.

En d'autres termes, l'algorithme identifie d'abord les chemins les plus courts formés d'un seul arc, ensuite il recherche ceux qui nécessitent

deux arcs et ainsi de suite jusqu'à n-1. Cette stratégie d'exploration dans un graphe porte le nom de recherche en largeur d'abord ou en anglais "breadth first search". On dit aussi que l'exploration du graphe se fait, niveau par niveau (se référer à la figure 3.7).

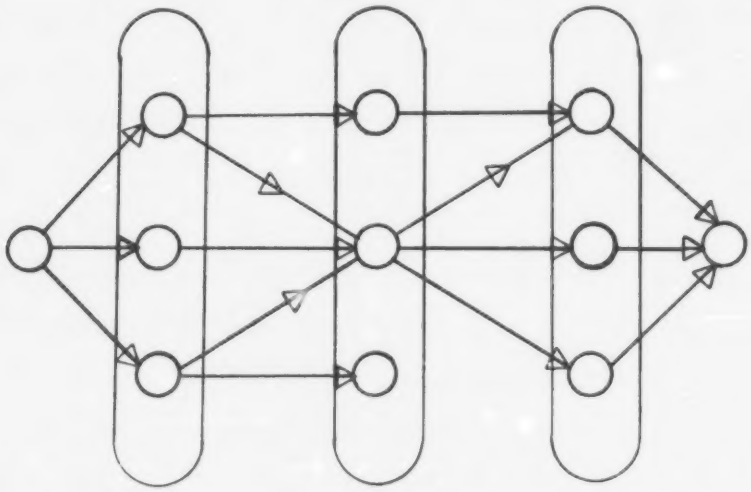


Figure 3.7 : Recherche en largeur d'abord

3.10 Recherche en profondeur d'abord

Contrairement à la spécialisation de l'algorithme du simplexe sur un réseau, l'algorithme général ne corrige pas immédiatement les étiquettes des descendants d'un sommet amélioré. Dans l'algorithme LQUEUE, par exemple, à partir d'un sommet i la visite s'effectue niveau par niveau. Dans le cas où Π_i vaut Π_i^*, cela crée un manque à gagner.

Une manière de parcourir immédiatement une sous-arborescence sans utiliser une structure de données compliquée est d'utiliser une pile. Une pile est une liste dans laquelle toute insertion ou suppression d'élément se fait à une extrémité, appelée le dessus de la pile. L'appellation anglo-saxone consacrée pour les piles est "LIFO list" ("last in, last out").

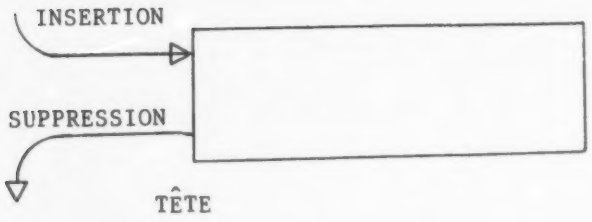


Figure 3.8 : Une pile

L'insertion des sommets améliorés selon la méthode LIFO correspond à une exploration du haut vers le bas et de droite à gauche en passant, s'il y a lieu, au dernier descendant de chaque sommet ("reverse thread"). On dit aussi que la fouille s'effectue en profondeur d'abord car nous suivons un chemin le plus loin possible avant de revenir en arrière.

Cette politique d'insertion-suppression des sommets n'est pas bien adaptée à la recherche des chemins minimaux. En effet, dans les premières itérations, au moment où les sommets reçoivent une amélioration pour la première fois, il est peu probable d'affecter aux sommets des étiquettes permanentes. Dans ce cas, il n'est pas avantageux de corriger immédiatement les marques de leurs descendants. En outre, Kershenbaum (1981) a montré que la complexité théorique d'un algorithme utilisant une pile est $O(n2^n)$. Pour ces raisons, en pratique nous n'utilisons pas un algorithme dont la liste est une pile. Par contre, en la couplant avec une autre structure de données nous pouvons obtenir des résultats intéressants. À ce sujet, nous donnons de plus amples détails à la section suivante.

3.11 Listes à doubles entrées

L'utilisation des listes à doubles entrées pour la recherche des chemins les plus courts vise à traiter différemment les sommets qui reçoivent leur étiquette la première fois des sommets déjà améliorés. D'Esopo (voir Pollack et Wiebenson, 1960) a été le premier à soulever cette idée. Elle consiste à traiter en priorité les sommets qui reçoivent une amélioration pour au moins la deuxième fois.

Une procédure possible de marquage est la suivante (Pape, 1974). Après k itérations nous faisons :

$A^k \leftarrow \{ i \in N \text{ tel que } \pi_i^k < \pi_i^{k-1} \}$ sommets améliorés à l'étape k.

$M^k \leftarrow M^{k-1} \cup A^k$ sommets améliorés jusqu'à maintenant.

$P^k \leftarrow A^k - \{A^k \cap M^{k-1}\}$ sommets qui reçoivent une amélioration pour la première fois à l'étape k.

$S^k \leftarrow A^k - P^k$ sommets qui reçoivent une amélioration pour au moins la seconde fois à l'étape k.

$H \leftarrow (H \cup P^k) - (H \cap S^k)$ ensemble des sommets qui ont été modifiés une seule fois.

Si $S^k \neq \emptyset$ alors $Q^k \leftarrow S^k$

Sinon si $H \neq \emptyset$ alors $Q^k \leftarrow H$; $H \leftarrow \emptyset$

Sinon terminer

$$\pi_j^{k+1} = \begin{cases} \min\limits_{j \in \Gamma_i} \{\pi_j^k, \pi_i^k + \ell_{ij}\} & \text{pour tout } i \in Q^k \\ \pi_j^k & j \notin \Gamma_i \quad Q^k \end{cases}$$

Après ceci, k est augmenté d'une unité et nous recommençons. L'algorithme termine lorsque aucune amélioration n'est possible.

Dans cette section, nous analysons deux variantes utilisant des listes à double entrées, soit les algorithmes de Pape (1974, 1980) et Pallottino (1979).

3.11.1 Algorithme LDEQUE

La mise en oeuvre faite par Pape (1974, 1980) de la suggestion de D'Esopo repose sur l'utilisation de deux listes. La première est une pile

dans laquelle sont insérés les sommets qui reçoivent une amélioration pour au moins la deuxième fois et la deuxième est une file qui emmagasine ceux pour qui l'étiquette est fixée pour la première occasion. Clairement, ces deux listes sont disjointes. Il est donc possible de les coupler. Cette nouvelle structure de données permet l'insertion aux deux extrémités de la liste tandis que l'élément ôté est pris à la tête. Dans la nomenclature informatique anglo-saxone, elle porte le nom "double-ended-queue" ou "deque" (Knuth, 1968). Pour cette raison nous désignons l'algorithme de Pape LDEQUE.

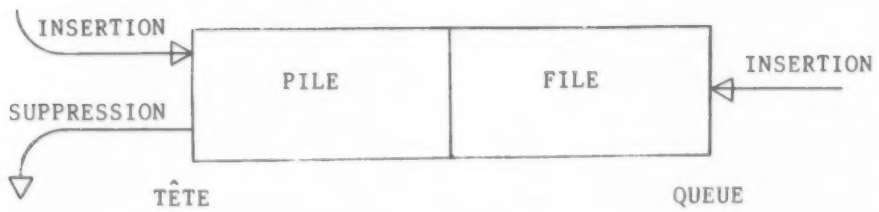


Figure 3.9 : Structure de données utilisée par Pape (1974)

Remarquons que chaque sommet est inséré une seule fois à la fin de cette liste à double entrées. Par conséquent, après un certain nombre d'itérations la méthode se comporte d'une façon identique à l'utilisation d'une pile. Donc, dans le pire des cas, la complexité de l'algorithme LDEQUE est $O(n2^n)$. Malgré cette piètre borne théorique, cette stratégie affiche de très bonnes performances lors des expérimentations numériques. La justification est la suivante. Lorsqu'un sommet i reçoit une amélioration pour au moins la deuxième fois, il est probable que la marque de ce sommet soit la longueur optimale. Alors, nous sommes intéressés à corriger immédiatement toutes les étiquettes des sommets de la sous-arborescence dont i est la racine. Cela est fait à l'aide d'une pile.

3.11.2 Algorithme L2QUE

La contribution de Pallottino est de remplacer la pile de la liste à doubles entrées précédente par une file. Dans ce cas, la visite de la sous-arborescence de racine i se fait niveau par niveau. La structure de données obtenue en joignant deux files est une double-file ou en anglais "2queue". Nous nommons l'algorithme engendré par cette structure de données L2QUE.

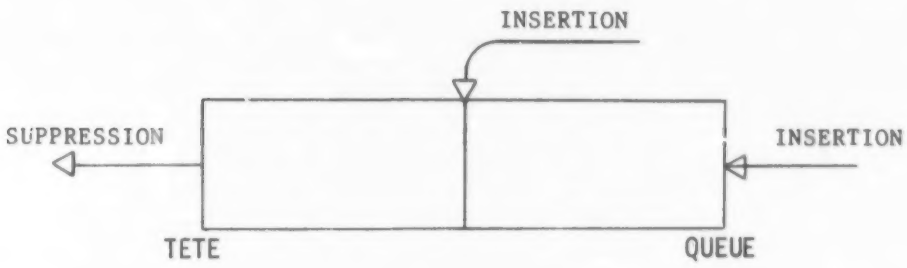


Figure 3.10 : Double-file utilisée par Pallottino

L'avantage d'utiliser cette structure de données est que lorsque toutes les étiquettes ont été révisées à la baisse, au moins une fois, l'algorithme se poursuit en appliquant la méthode de Moore. Au pire des cas, un sommet est sélectionné $O(n^2)$ fois, donc $\tilde{c}$ vaut n^2. Or, la complexité des opérations d'insertion et de suppression des sommets dans cette structure de données est $O(1)$. Alors la complexité de l'algorithme L2QUE est $O(n^2m)$ (Pallottino, 1984).

3.12 Listes partitionnées - Fonction Seuil

La motivation qui a suscité cette nouvelle stratégie est l'intégration des propriétés des algorithmes par recherche du minimum d'abord et les performances des algorithmes avec des listes à doubles entrées. Cette motivation est d'autant plus stimulée que les algorithmes ayant une bonne borne théorique ne sont pas toujours plus rapides que les autres lors des

expérimentations numériques. Selon une étude indépendante (Denardo et Fox, 1979), aucun algorithme n'a pu être identifé le meilleur sur toutes les topologies de réseaux testés. Il semble, selon d'autres résultats (Dial et al., 1979; Gallo et Pallottino, 1988) que les méthodes LDEQUE et L2QUE se distinguent des autres méthodes pour les réseaux peu denses, tandis que l'algorithme SDIAL est meilleur pour les problèmes denses. Plus particulièrement, les travaux récents de Glover et al. (1984, 1985) ont porté sur la mise en oeuvre de nouveaux algorithmes alliant la rapidité de calcul des algorithmes LDEQUE ou L2QUE et la bonne borne théorique des algorithmes par recherche du minimum d'abord. En premier lieu, ces auteurs ont développé un algorithme général facilitant les études théoriques. Cet autre cadre unificateur utilise le concept de partionnement. Selon ces auteurs, cette méthode générale constitue un lien conceptuel entre les algorithmes par ajustement progressif et ceux par extension sélective, les incitant à créer une méthode hybride unissant les avantages des deux. Cela est accompli avec une fonction Seuil.

Nous commençons par exposer l'algorithme de partitionnement. Ensuite, nous énonçons quelques résultats fondamentaux le concernant. Après quelques remarques, nous terminons ce chapitre en présentant des implantations avec la fonction Seuil.

3.12.1 Algorithme de partitionnement

ALGORITHME 5 (Glover, Klingman, Phillips, 1982)

1. Initialisation

$$\Pi_i \leftarrow \begin{cases} 0 & \text{si } i=r \\ \text{infini} & \text{sinon} \end{cases}$$

$$p_i \leftarrow \begin{cases} -1 & \text{si } i=r \\ r & \text{sinon} \end{cases}$$

$k \leftarrow 0$ compteur d'itérations

NOW $\leftarrow \{r\}$, NEXT $\leftarrow \emptyset$

2. Sélection

Si NOW = $\emptyset$ aller à l'étape 4

Sinon, choisir un sommet i quelconque de NOW; NOW ← NOW - {i}

3. Amélioration

Pour chaque $(i,j) \in A_i$ faire

si $\Pi_i - \Pi_j + \ell_{ij} < 0$ alors faire

début

$\Pi_j \leftarrow \Pi_i + \ell_{ij}$

$p_j \leftarrow i$

Si $j \notin$ NOW et si $j \notin$ NEXT alors ajouter j à NEXT

fin

fin

Aller à l'étape 2.

4. Transfert

Si NEXT = $\emptyset$ alors terminer

Sinon poser k ← k+1 et transférer tous les sommets de NEXT à NOW

aller à l'étape 2.

■

Glover et al. (1985) ont démontré :

Lemme 3.2

À l'étape 4 de la $k^{ième}$ itération, chaque sommet j de l'ensemble NEXT vérifie les deux propriétés :

- le sommet j est à au moins k arcs de la racine[1],
- il existe au moins un chemin le plus court allant de la racine à ce sommet ayant au moins k arcs.

---

[1] Dans ce cas, on parle de profondeur d'un sommet. C'est la longueur en nombre d'arcs, dans un arborescence, du chemin allant de la racine à ce sommet.

Lemme 3.3

À l'étape 4 de chaque itération, au moins un sommet de NEXT ne revient pas dans NEXT. Son étiquette possède la valeur minimale.

Pour notre part, nous nous contentons de justifier ces lemmes en remarquant que le transfert des sommets de NEXT à NOW à l'étape 4 revient à créer une suite $NOW^0, NOW^1, \ldots$ assimilable aux ensembles $Q^0, Q^1, \ldots$ énoncés auparavant. En d'autres mots, c'est dire qu'après avoir analysé k niveaux du graphe, tous les sommets de l'ensemble $NEXT^k$, sauf au moins un, n'ont pas reçu leur étiquette permanente. Contrairement aux algorithmes LQUEUE, LDEQUE et L2QUE où le partitionnement de Q est implicite, l'algorithme 5 le rend explicite. En outre, il est bien connu (Lawler, 1976; Yen, 1970) que l'analyse des sommets à améliorer doit se faire parmi ceux pour lesquels les prédécesseurs ont déjà été modifiés et l'algorithme 3 s'applique à le faire. Par contre, l'originalité de la procédure de partitionnement est d'identifier clairement ces sommets à chaque itération.

Glover et al. (1984) soutiennent que la complexité de la procédure de partitionnement est O(mn). Cela dépend du lemme suivant :

Lemme 3.4

La complexité théorique de l'algorithme 5 est O(mn) si le transfert des sommets de NEXT à NOW nécessite O(1) calcul et $O(mn^2)$ sinon.

Preuve Étant donné que l'exploration du graphe s'effectue niveau par niveau (conséquence du partitionnement), à chaque itération, au moins un sommet reçoit une étiquette permanente. Donc k est borné par n-1. De plus, à chaque itération l'algorithme 5 analyse au plus m arcs. Par conséquent, la complexité théorique de l'algorithme 5 est O(mn) si le transfert des sommets de NEXT à NOW consiste à renommer l'ensemble NEXT par NOW. Autrement, si à l'étape 4, le transfert requiert le balayage des éléments de l'ensemble NEXT; alors la procédure est $O(mn^2)$.

Du point de vue pratique, il est tout à fait inutile de mettre en oeuvre sur un ordinateur cette méthode. En revanche, elle est le tremplin vers des algorithmes nouveaux, notamment, si à l'étape 4 on ne transfère pas tous les sommets.

3.12.2 Fonctions avec Seuil

Lorsque les longueurs des arcs sont sujettes à la non-négativité, nous démontrons le lemme suivant :

Lemme 3.5

Soit $R = (N,U,L)$ tel que $\ell_u > 0$ pour tout $u \in U$. Posons $d^k = \underset{i \in NEXT}{\text{Min}} \{\Pi_i\}$ à l'itération k. Alors la suite $d^0, d^1, \ldots$ est monotone non décroissante.

Preuve Soit $NEXT^k$, l'ensemble des sommets qui sont dans NEXT à l'étape 4 de l'itération k, avant le transfert dans NOW. Au début de l'itération k+1, NOW est constitué des sommets de $NEXT^k$ et tous les sommets améliorés à l'itération k+1 sont placés dans $NEXT^{k+1}$. Par conséquent, nous avons

$$d^{k+1} = d^k + \underset{i \in NEXT^k, j \in NEXT^{k+1}}{\text{Min}} \{\ell_{ij}\} .$$

Sous l'hypothèse de non-négativité des arcs, d^{k+1} est nécessairement plus grand ou égale à d^k.

∎

Un corollaire immédiat concerne les sommets qui reçoivent une étiquette permanente à l'itération k.

Corollaire 1

Tous les sommets i dans $NEXT^{k-1} \cup NEXT^k$ dont l'étiquette satisfait l'inégalité $d^{k-1} < \Pi_i < d^k$, $k > 1$, sont tels que $\Pi_i = \Pi_i^*$.

Ces considérations nous assurent que l'algorithme obtenu en substituant l'étape 4 de l'algorithme de partitionnement par l'étape 4A possède aussi une borne théorique bornée par un polynôme lorsque les longueurs des arcs sont non négatives.

Étape 4A - Transfert

Si NEXT = Ø alors terminer

Sinon poser k ← k+1 et transférer un sous-ensemble, incluant au moins un sommet i pour lequel $\Pi_i = \underset{j \in NEXT}{Min} \{\Pi_j\}$, de NEXT.

Aller à l'étape 1.

Il convient de préciser que les sommets qui ne sont pas tranférés demeurent dans NEXT et que les sommets améliorés à l'itération suivante s'ajoutent à ces sommets. D'autre part, il est intéressant de noter que si l'on transfère seulement les sommets dont l'étiquette est la plus faible, alors l'algorithme de partitionnement devient une implantation de la méthode de Dijkstra. Une façon possible de procéder afin d'éviter le calcul du minimum sur tous les éléments de NEXT est de faire passer les c-1 premiers sommets, pour un c fixé, à l'ensemble NOW et de rechercher le sommet dont l'étiquette est la plus faible parmi les sommets restants. Ce dernier est ensuite transféré.

Finalement, il semble qu'il est préférable de faire passer tous les sommets de NEXT à NOW dont l'étiquette est inférieure ou égale à une valeur Seuil. Bien entendu, à l'itération k, cette valeur Seuil, t_k,[1] doit être plus grande ou égale à d^k. Les auteurs de ces travaux, Glover et al. (1984), justifient cet état de fait en mentionnant qu'à chaque itération, plus d'un sommet reçoit son étiquette permanente. Ceci réduit la taille de l'ensemble NEXT et compense l'effort déployé à son balayage.

---

[1] En anglais : "threshold value".

Plusieurs expérimentations ont été menées. En premier lieu, plusieurs fonctions avec Seuil ont été construites. Par exemple, la fonction t ne nécessite pas toujours le parcours de tout l'ensemble NEXT. Par une méthode plus efficace, elle est mise à jour en adaptant les paramètres qui la composent. Au détriment de cette méthode, nous soulignons que la calibration de ces paramètres reposent sur une connaissance a priori du réseau. Notamment, Divoky (1987) a élaboré une fonction t qui prend avantage que la distribution des longueurs des arcs est exponentielle. Finalement, d'autres stratégies d'insertion des sommets ont été mises en oeuvre. En particulier, Glover et al. (1985b, voir algorithme "THRESHX1") suggèrent l'utilisation d'une troisième liste afin de réduire encore plus la cardinalité de l'ensemble NEXT. À l'étape 2 de cet algorithme, les sommets dont la marque est inférieure à la valeur Seuil sont insérés dans une liste NOW' au lieu d'être placés dans NEXT. Quand NOW est vide, les sommets de NOW' sont transférés dans NOW, si NOW' est non vide. Sinon, la valeur t est mise à jour et des sommets de NEXT à NOW sont transférés. Nous croyons que cela est superflu car nous constatons qu'à l'étape 2 il est préférable de placer les sommets dont l'étiquette est en-dessous du seuil directement dans NOW. Nous nommons cet algorithme, LTHRS, et voici les détails :

ALGORITHME LTHRS

1. Initialisation

$$\Pi_i \leftarrow \begin{cases} 0 & \text{si } i=r \\ \text{infini} & \text{sinon} \end{cases}$$

$$p_i \leftarrow \begin{cases} -1 & \text{si } i=r \\ r & \text{sinon} \end{cases}$$

$k \leftarrow 0$, $t^k \leftarrow 0$

NOW $\leftarrow \{r\}$, NEXT $\leftarrow \emptyset$

2. Sélection

Si NOW = $\emptyset$ aller à l'étape 4

Sinon, choisir un sommet i quelconque de NOW; NOW ← NOW - {i}

3. Amélioration

Pour chaque $(i,j) \in \Lambda_i$ faire

si $\Pi_i - \Pi_j + \ell_{ij} < 0$ alors

début

$\Pi_j \leftarrow \Pi_i + \ell_{ij}$

$p_j \leftarrow i$

si $\Pi_i < t^k$ alors NOW ← NOW U {j}

sinon NEXT ← NEXT U {j}

fin

fin

Aller à l'étape 2.

4. Transfert

Si NEXT = $\emptyset$ alors STOP

Sinon, déterminer t^{k+1}, transférer les sommets i dans NEXT

tel que $\Pi_i < t^{k+1}$.

k ← k+1, aller à 2.

La complexité de cet algorithme est $O(n^2 m)$ car nous devons balayer tous les éléments de l'ensemble NEXT afin de mettre à jour t. Nous faisons de même afin de transférer des sommets de cet ensemble dans NOW. Le lecteur intéressé à connaître les détails liés aux différentes fonctions t est prié de se référer aux articles de Glover et al. (1984, 1985) et Divoky (1987). Nous décrivons la mise en oeuvre informatique de cet algorithme au chapitre suivant.

CHAPITRE 4

MISE EN OEUVRE ET EXPÉRIMENTATIONS NUMÉRIQUES

Dans ce chapitre, nous traitons les huit algorithmes du chemin le plus court, présentés au chapitre précédent, du point de vue de leur mise en oeuvre informatique.

Pour faire suite au choix du langage de programmation pour l'implantation de l'outil TACTIC (voir chapitre 7), nous présentons une analyse détaillée des performances de ces algorithmes lorsqu'ils sont programmés en langage C. À notre connaissance, c'est la première fois qu'une telle analyse est effectuée. Nous soulignons que les résultats d'expérimentations numériques décrit dans la littérature scientifique proviennent pour la plupart de programmes écrits en FORTRAN (e.g. Gallo et Pallottino, 1988; Glover et al., 1985; Dial et al., 1979; etc.). Mentionnons que Denardo et Fox (1979) et Sauvageau (1978) discutent de l'implantation de quelques algorithmes de recherche des chemins minimaux en langage assembleur (COMPASS) de l'ordinateur CDC Cyber de l'Université de Montréal.

Le recours aux expérimentations numériques est un moyen très employé dans la littérature scientifique pour juger l'efficacité des algorithmes du chemin le plus court. À ce sujet, rappelons les travaux importants de Gallo et Pallottino (1988), Latour (1987), Glover et al., (1985a, 1985b), Denardo et Fox (1979) et Dial et al. (1979).

La motivation du travail présenté dans ce chapitre provient du fait que les analyses de la complexité théorique ne tiennent pas compte que le temps pris par certaines opérations tels les accès aux données d'un tableau, l'incrémentation des variables qui servent d'indice et les comparaisons dépend du langage de programmation. Par exemple, c'est le cas du langage C où l'utilisation adéquate des pointeurs rend plus efficace l'accès aux données d'un tableau.

L'objectif que nous poursuivons dans ce chapitre consiste à analyser les performances des algorithmes de recherche des chemins minimaux dans un réseau lorsque l'implantation prend avantage des caractéristiques particulières du langage C. Ainsi, nous pouvons vérifier si les résultats obtenus à l'aide de programmes efficients Fortran s'appliquent quand ces mêmes algorithmes sont aussi implantés efficacement en langage C.

En premier lieu, nous rappelons les techniques traditionnelles utilisées pour l'implantation des algorithmes du chemin le plus court. Mentionnons que la notation que nous employons pour cette présentation est fortement inspirée de celle utilisée par Gallo et Pallottino (1988). Par la suite, nous donnons le détail des implantations de huit algorithmes, présentées au chapitre précédent, en langage C. À notre connaissance, il n'existe aucun résultat d'analyse numérique dans la littérature scientifique pour l'algorithme SRHEAP dû à Ahuja et al. (1988). Dans ce chapitre, nous pallions à cette lacune en rapportant plusieurs résultats expérimentaux. Finalement, nous discutons des impacts de la programmation en langage C sur l'efficacité des algorithmes du chemin le plus court.

4.1 Mise en oeuvre informatique d'un algorithme du chemin le plus court

Pour emmagasiner l'information relative à un graphe, un certain nombre de structures de données peuvent être utilisées (Gondran et Minoux, 1985). En pratique, elle ne sont généralement pas équivalentes du point de vue de l'efficacité des algorithmes. En ce qui nous concerne, la représentation la mieux adaptée aux algorithmes de recherche des chemins minimaux est basée sur la matrice d'incidence sommets-sommets (Dial et al., 1979; Aho et al., 1987; Gallo et Pallottino, 1988).

La matrice d'incidence sommets-sommets, ou matrice d'adjacence, d'un graphe G=(N,U) sans arc parallèle et comportant n sommets est une matrice $A=(a_{ij})$ de dimension nxn, à coefficients 0 ou 1, où chaque rangée correspond à un sommet de G, où chaque colonne correspond à un sommet de G, et où

$$a_{ij} = \begin{cases} 1 & \text{si et seulement si } (i,j) \in U, \\ 0 & \text{sinon} \end{cases}$$

Nous pouvons encore signaler une représentation intimement liée à la précédente, appelée matrice pondérée d'adjacence. Dans une telle matrice, l'élément a_{ij} est égal à la longueur de l'arc (i,j). Si un tel arc est absent du graphe, alors a_{ij} contient une valeur spéciale (e.g. très grande) qui n'est pas susceptible d'être trouvée dans l'ensemble des longueurs des arcs. Dans le cas d'un graphe peu dense, il y a un gaspillage important d'espace mémoire. Pour éliminer cet inconvénient, il est préférable de décrire seulement les termes non nuls de la matrice. Cela revient à représenter G par la liste des successeurs de ses sommets, en anglais "forward star form". La liste des successeurs relative au sommet i est une liste, dans un ordre arbitraire, de tous les sommets adjacents à i. La figure 4.1 montre la représentation par listes des successeurs d'un graphe orienté dans laquelle nous utilisons des listes simplement chaînées.

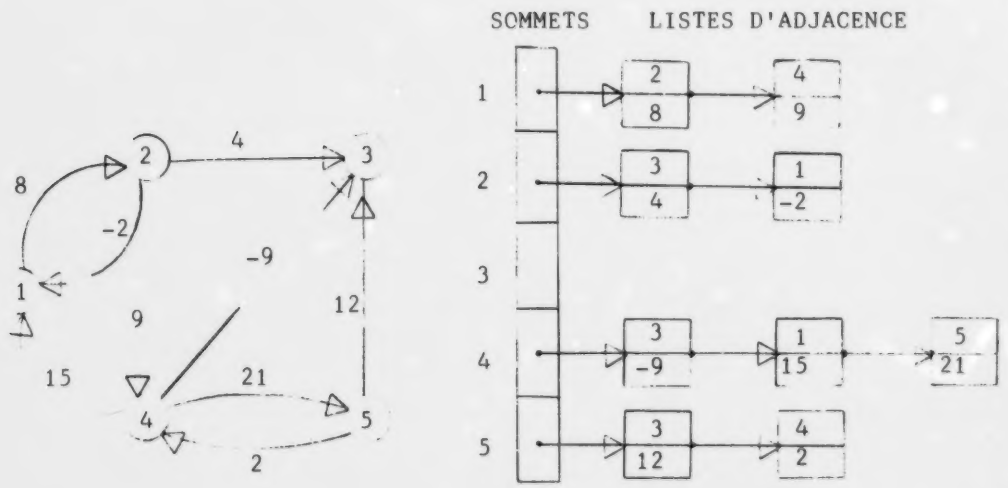


Figure 4.1 : G représenté par la liste des successeurs. Lorsque les arcs sont munis de longueurs, celles-ci apparaissent dans les cellules des listes chaînées.

Si l'on tient compte de la longueur des arcs, cette structure de données requiert n+3m mots mémoire. D'un autre côté, si le graphe ne change pas durant l'application (c'est le cas pour notre problème), il est possible de représenter G dans la mémoire d'un ordinateur avec seulement n+2m+1 mots mémoire. C'est une économie de m mots mémoire. Pour cela, nous utilisons deux tableaux A[] et ND[] de dimension respective n+1 et m. Pour chaque sommet i, la liste des successeurs de i est contenue dans le vecteur ND[] à partir de la case A[i] jusqu'à la case A[i+1]-1. Par convention, nous posons A[i]=A[i+1] quand la liste des successeurs du sommet i est vide. Enfin, nous avons A[n+1]=m+1. Pour conserver la longueur des arcs, nous réquisitionnons un tableau supplémentaire LNGTH[] aussi de dimension m. La figure 4.2 illustre cette technique.

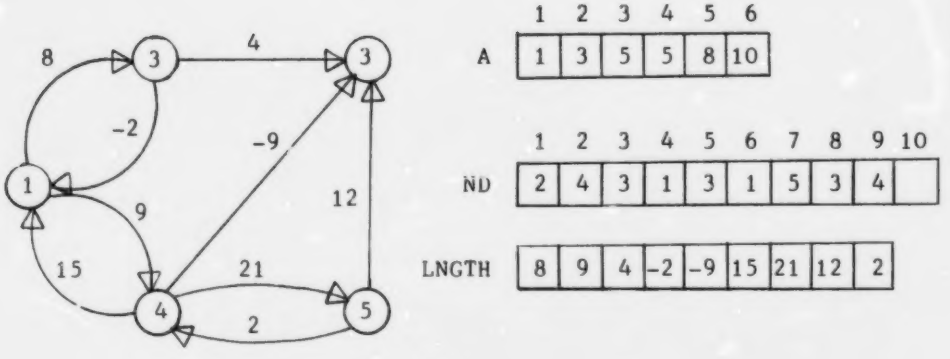


Figure 4.2 : G représenté en machine avec les tableaux A[], ND[] et LNGTH[].

Cette représentation du graphe en machine, bien connue dans la littérature scientifique (Dial et al., 1979; Gondran et Minoux, 1985; Gallo et Pallottino, 1988) requiert 2m+n+1 mots mémoire et elle permet de balayer très efficacement tous les arcs de l'ensemble A_i en exécutant :

```
START ← A[i]
END   ← A[i+1]
for K=START; K<END; step 1 do
   begin
      j ← ND[K]
      LNGT ← LNGTH[K] /* longueur de l'arc (i,j) */
      .
      . /* traitement de l'arc (i,j) */
      .
   end
```

Ensuite, pour décrire l'arborescence des chemins les plus courts, nous utilisons deux vecteurs, D[] et P[], de dimension n. À la fin de l'exécution de l'algorithme, D[i] vaut la longueur minimale pour se rendre de r à i et P[i] est le dernier sommet qui a contribué à améliorer i. Dans ce cas, le chemin minimal allant de r à i, déterminé par l'algorithme est

$$h^*_{ri} = (r,\ldots,P[P[i]],P[i],i).$$

Initialement, nous posons

$$D[i] = \begin{cases} 0 & \text{si } i=r \\ \text{HUGE} & \text{sinon, où HUGE est une valeur très grande} \end{cases}$$

$$P[i] = \begin{cases} 0 & \text{si } i=r \\ r & \text{sinon} \end{cases}$$

Finalement, la procédure générale (voir section 3.7, Approche unificatrice) pour la détermination des chemins les plus courts issus d'un

sommet r, qui tire avantage du fait que l'on puisse accéder rapidement aux successeurs d'un sommet avec les tableaux A[] et ND[], est implantée efficacement de la manière suivante (voir Gallo et Pallottino, 1988) :

Procédure 4.1 : Chemins les plus courts issus du sommet r
(où r est le numéro du sommet racine)

```
 1  begin
 2     integer : i,j
 3     integer : START,END,K
 4     integer : DJ
 5
 6     INIT (Q,r)
 7     do
 8       begin
 9          QOUT (i)
10          START ← A[i]
11          END ← A[i+1]
12          for K=START; K<END; step 1 do
13             begin
14                j ← ND[K]
15                DJ ← D[i] + LNGTH[K]
16                if DJ<D[j] then
17                   begin
18                      D[j] ← DJ
19                      P[j] ← i
20                      QIN(j)
21                   end
22             end
23     while (Q≠Ø)
24  end.
```

Pour l'instant, les procédures INIT(), QOUT() et QIN() ne sont pas bien définies puisqu'elles dépendent de la structure de données choisie pour représenter la liste des noeuds candidats, Q. Nous donnons le détail en langage C de ces procédures pour chacun des huit algorithmes que nous décrivons plus loin, dans ce chapitre.

4.2 Implantation en langage C

Certains auteurs (e.g. Dial et al., 1979; Gallo et Pallottino, 1988) mentionnent que A[i] est un pointeur sur le tableau ND[]. C'est un abus de langage car A[i] est un entier. Un **pointeur** est une variable qui contient l'adresse d'une autre variable. De préférence (voir Aho et al., 1987), on dit que A[i] est un curseur. Un **curseur**, ou faux pointeur, est une variable, à valeur entière qui réfère à une case d'un tableau. Quoique le langage de programmation Pascal possède le type "pointeur", il n'est pas permis aux pointeurs de pointer sur les cases d'un tableau. Pour ce langage et pour les langages de programmation comme FORTRAN, Basic ou Algol, qui ne possèdent pas de type "pointeur" explicite, l'implantation la plus efficace d'un algorithme du chemin le plus court consiste à utiliser la procédure 4.1. Une excellente référence à ce sujet est Gallo et Pallottino (1988). L'article contient plusieurs programmes en pseudo-code et en FORTRAN et il est accompagné d'une disquette.

En langage C, beaucoup de relations existent entre les pointeurs et les tableaux. En effet, toute opération utilisant un indice de tableau peut aussi être réalisé grâce à des pointeurs. Or, la version utilisant des pointeurs est en général plus rapide. En fait, un renvoi à un tableau est toujours transformé par le compilateur en la création d'un pointeur indiquant le début du tableau. En conséquence, le nom même du tableau devient un pointeur. Contrairement au langage Pascal, ceci permet de rendre plus efficace la procédure 4.1 en remplaçant le tableau d'entier A[] par un

tableau de vrais pointeurs. De plus, le fait que les tableaux ND[] et LNGTH[] sont parallèles nous invite à considérer une autre organisation des données. La déclaration suivante en langage C :

```
struct NODE {
             struct ARC *a;    /*tableau A[]*/
            }node [n+2];

struct ARC  {
             struct NODE *nd; /*tableau ND[]/
             int lngth;       /*tableau LNGTH[]*/
            }arc [n+1];
```

définit deux tableaux dont les éléments sont des structures. Nous soulignons que les indices des tableaux commencent toujours à zéro en langage C et non pas à un comme en FORTRAN. Au lieu de renommer les sommets {0,1,...,n-1}, par souci de cohérence, nous préférons perdre les cases node[0] et arc[0].

Bien que la déclaration

```
struct ARC *a[ n+2] ;
```

est équivalente à la déclaration[1]

```
struct NODE {struct ARC *a;
            }node [n+2];
```

nous adoptons cette dernière dans le but de regrouper d'autres tableaux parallèles dont les indices sont les numéros des sommets. Pour l'affichage graphique, par exemple, nous utilisons les tableaux X[] et Y[] qui contiennent les coordonnées des noeuds tandis que les algorithmes de recherche des chemins minimaux emploient les tableaux D[] et P[]. L'initialisation de cette structure de données s'effectue par les instructions

```
i=1;
while (i<=n+1) {scanf ("%d", &j);
                node [i].a = *(arc+j); /*&arc[j]*/
               }
```

---

[1] Cela est vrai car la structure NODE n'a qu'un seul membre qui est un pointeur sur la structure ARC. Dans le cas général, un tableau de pointeurs sur des structures n'est pas équivalent à un tableau de structures.

```
i=1;
while (i<=m)   {scanf ("%d %d", &j, &lngth);
                arc [i].nd = *(node+j);
                arc [i].lngth = lngth;
```

quand le réseau est stocké par la liste des successeurs.

Finalement, nous insérons les tableaux parallèles P[] et D[] dans la structure "NODE" de la façon suivante :

```
struct NODE {
             struct ARC *a;
             struct NODE *p; /*tableau P[]*/
             int d; /*tableau D[]*/
            }node [n+2];
```

alors, en langage C, la procédure 4.1 prend la forme suivante :

Procédure 4.2 : Chemins les plus courts issus du sommet r
(où r est un pointeur qui contient l'adresse du sommet origine : i.e. struct NODE *r;)

```
 1 {
 2   struct NODE *i, *j;
 3   struct ARC *start, *end, *k;
 4   int dj;
 5
 6   INIT (Q,r)
 7   do
 8     {
 9       QOUT (i)
10       start = i + a;
11       end = (i+1) + a;
12       for (k=start; k<end; k++)
13         {
14           j = k + nd;
15           dj = i + d + k + lngth;
```

```
16          if (dj<j → d)
17            {
18              j → d = dj;
19              j → p = i;
20              QIN(j)
21            }
22        }
23      }while (Q≠Ø)
24  }
```

Soulignons qu'en plus d'améliorer les performances des algorithmes du chemin le plus court (à ce sujet voir section 4.8), cette implantation est une adaptation en langage C d'une approche de programmation qui est reconnue pour son efficacité dans la littérature scientifique (e.g. Dial et al., 1979; Gallo et Pallottino, 1988). D'ailleurs, nous faisons remarquer qu'il y a une correspondance biunivoque entre les lignes de la procédure 4.2 et les lignes de la procédure 4.1.

Nous avons aussi exploré d'autres implantations possibles. Par exemple, nous avons considéré l'utilisation de l'allocation dynamique de la mémoire pour la représentation de la liste Q en machine. Des tests préliminaires ont indiqué que cette approche est infructueuse. Bien que cela soit très élégant, l'inefficacité est liée au temps non négligeable pris par l'allocateur de mémoire. De plus, mentionnons que l'implantation d'un dispositif d'allocation dynamique de la mémoire dépend du compilateur et de l'ordinateur. Dans ce cas, l'efficacité des programmes qui utilisent ce dispositif varie beaucoup selon les ordinateurs. Par conséquent, nous déconseillons cette approche pour une mise en oeuvre efficiente des algorithmes du chemin le plus court.

Pour la suite de notre exposé, nous avons choisi huit structures de données pour implanter la liste Q. Les quatre premières structures de données sont des listes et elles donnent lieu à des algorithmes de marquage par ajustements progressifs. Il s'agit des algorithmes LQUEUE, LDEQUE, L2QUE et LTHRS. Les quatre autres sont des implantations d'une file de priorité et elles correspondent aux algorithmes par ajustements sélectifs SDKSTR, SHEAP, SDIAL et SRHEAP. Pour chacun de ces algorithmes, nous définissons les structures et les variables utilisées et nous montrons le détail des procédures INIT(), QIN() et QOUT() en langage C. Enfin, nous rappelons que pour tous ces algorithmes, l'organisation des données pour conserver les arcs demeure la même, soit :

```
struct ARC  {
             struct NODE *nd; /*tableau ND[]*/
             int lngth;       /*tableau LNGTH[]*/
            }arc [m+1];
```

4.3 Algorithmes LQUEUE, LDEQUE et L2QUE

Dans cette section, nous traitons les trois algorithmes LQUEUE, LDEQUE et L2QUE simultanément car leur implantation efficace (voir Gallo et Pallottino, 1988) se fait à l'aide d'une liste simplement chaînée.

Lorsque les éléments de la liste correspondent aux entiers 1,2,...,n (c'est notre cas), une implantation efficiente consiste à utiliser un tableau Q[], dont les indices sont les numéros des sommets, de dimension n+1, et une variable entière LAST qui indique la fin de la liste. Les éléments de ce tableau sont des faux-pointeurs qui sont définis de la façon suivante :

$$Q[i] = \begin{cases} j & \text{si } i \text{ est dans la liste et } j \text{ est l'élément suivant,} \\ 0 & \text{si } i \text{ n'est pas dans la liste,} \\ n+1 & \text{si } i \text{ est le dernier élément de la liste } (i=\text{LAST}). \end{cases}$$

De plus, pour faciliter l'insertion et la suppression des éléments de la liste, celle-ci est organisée comme une liste circulaire, c'est-à-dire que la case n+1 du tableau Q[] contient le numéro du premier sommet. La figure 4.3 montre le contenu du tableau Q[] quand la liste des sommets est {2,5,9,7,4}. Mentionnons qu'avec cette implantation d'une liste chaînée, les réponses aux questions "est-ce que j est dans le liste Q?" et "la liste Q est-elle vide?" s'obtiennent respectivement d'une façon triviale par les deux instructions

```
if Q[j] = 0 then ... /*j n'est pas dans Q*/
if Q[n+1] = n+1 then ... /*Q est vide*/.
```

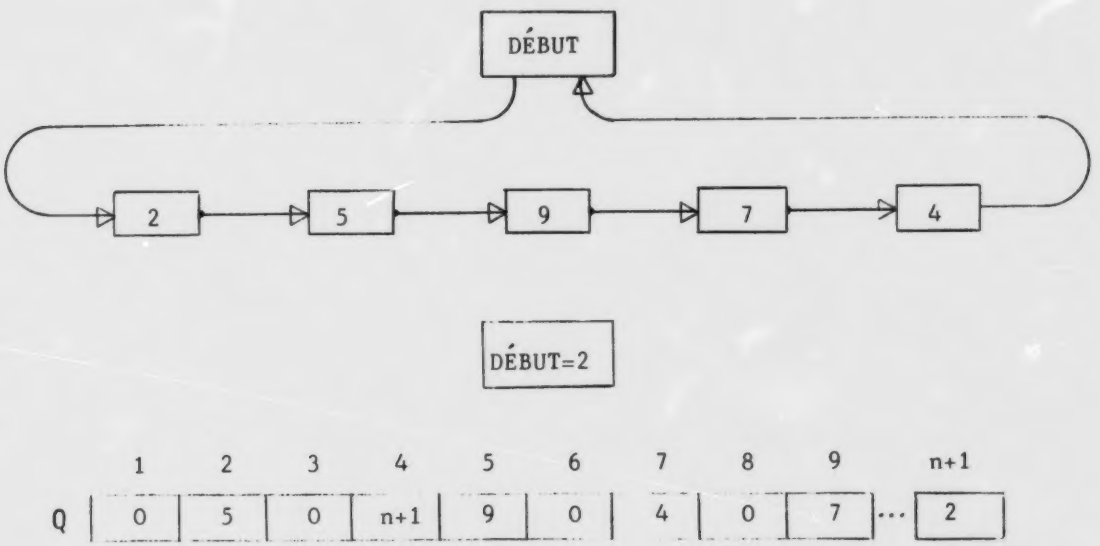


Figure 4.3 : Une liste simplement chaînée implantée avec le tableau Q[].

En langage C, nous prenons avantage que les éléments du tableau Q[] peuvent être de véritables pointeurs. Ainsi, nous redéfinissons la structure "NODE" de la façon suivante :

```
struct NODE {
            struct ARC *a;  /*tableau A[]*/
            struct NODE *p; /*tableau P[]*/
            int d;          /*tableau D[]*/
            struct NODE *q; /*tableau Q[]*/
           } node [n+2];
```

Avec cette structure de données, les procédures INIT() et QOUT() qui sont communes aux trois algorithmes sont les suivantes : (mentionnons que pour des raisons d'efficacité, nos programmes ne font pas d'appel de fonction, le code étant directement inséré dans la procédure 4.2. Pour cette raison, nous omettons les entêtes des fonctions et nous assumons qu'une fois définies, les variables sont globales. Les programmes sont disponibles sur disquette. Les commentaires indiquent l'équivalent en pseudo-code de la procédure 4.1).

Procédure INIT (Q,r) (LQUEUE, LDEQUE, L2QUE)

```
struct NODE *ptr, *n1, *last;
struct NODE *pntr;      /*uniquement pour l'algorithme L2QUE*/
                        /*le rôle de pntr est expliqué plus loin*/
ptr = (node+1); n1 = (node+n+1) /*voir structure NODE ci-haut*/
while (ptr < n1)
   {
    ptr -> p = r;       /*P[i]=r*/
    ptr -> d = HUGE;    /*D[i]=HUGE*/
    ptr -> q = '\000';  /*Q[i]=0, '\000' est le pointeur nul*/
    ptr ++;
   }
r -> p = '\000';    /*P[r]=0*/     r -> d = 0; /*D[r]=0*/
r -> q = n1;        /*Q[r]=n+1*/
n1 -> q = last = r; /*Q[n+1]=LAST=r*/
pntr = n1;          /*uniquement pour l'algorithme L2QUE*/
```

Procédure QOUT (i) (LQUEUE, LDEQUE, L2QUE)

```
i = n1 → q;              /*i=Q[n+1]*/
n1 → q = i → q;          /*Q[n+1]=Q[i]*/
i → q = '\000';          /*Q[i]=0*/
if (last==i) last=n1;
if (pntr==i) pntr=n1;  /*uniquement pour l'algorithme L2QUE*/
```

L'algorithme LQUEUE emploie une file pour maintenir la liste des sommets candidats. Ainsi, les sommets sont toujours insérés à la fin de la liste chaînée avec la procédure QIN() suivante :

Procédure QIN (j) (algorithme LQUEUE)

```
if (!j → q)              /*if Q[j]!=0*/
   {
    last → q = j;        /*Q[LAST]=j*/
    j → q = n1;          /*Q[j]=n+1*/
    last = j;            /*LAST=j*/
   }
```

Les algorithmes LDEQUE et L2QUE font un partitionnement implicite de la liste des sommets candidats afin de différencier les sommets qui reçoivent une amélioration pour la première fois des autres. Dans l'algorithme LDEQUE, les sommets améliorés pour au moins la seconde fois sont placés au début d'une pile, Q_1, et les autres sont insérés à la fin d'une file, Q_2. Puisque $Q_1 \cap Q_2 = \emptyset$, nous implantons cette structure de données, appelée "deque", avec une liste chaînée. L'insertion des sommets dans la liste s'accomplit avec la procédure QIN(j) que nous décrivons ci-bas. Notons que nous reconnaissons un sommet qui reçoit une amélioration pour la première fois en vérifiant la valeur de son étiquette.

Procédure QIN (j) (algorithme LDEQUE)

```
if (!j → q)
   {
    if (j → d < HUGE)  /*j a-t-il déjà reçu une amélioration*/
     {                 /*insertion au début de la liste*/
      j → q = n1 → q;  /*Q[j]=Q[n+1]*/
      n1 → q = j;      /*Q[n+1]=j*/
      if (last==n1) last=j;
     }
```

```
else
 {                    /*j est inséré à la fin de la liste*/
  last → q = j;
  j → q = nl;     /*comme algorithme LQUEUE*/
    last = j;
 }
```

Pallottino (1979) a mis au point une seconde implantation de la suggestion de d'Esopo en substituant la pile Q_1 qui reçoit les sommets améliorés par une file Q'_1. La condition $Q'_1 \cap Q_2 = \emptyset$ demeure, et la liste obtenue par la fusion de Q'_1 et Q_2 porte le nom "2queue". Du point de vue informatique, nous employons le même tableau Q[] que précédemment, mais nous utilisons aussi une variable supplémentaire, pntr, qui indique le point de jonction entre les deux files. Ainsi, la procédure QIN() pour l'algorithme L2QUE est la suivante :

Procédure QIN (j) (algorithme L2QUE)

```
if (!j → q)
   {
    if (j → d < HUGE)
     {                    /*insertion à la fin de la file Q'1*/
      j → q = pntr → q; /*Q[j]=Q[PNTR]*/
      pntr → q = j;     /*Q[PNTR]=j*/
      if (last==pntr) last=j;
      pntr = j;
     }
    else
     {                    /*insertion à la fin de la file Q2*/
      last → q = j;     /*comme précédemment*/
      j → q = nl;
      last = j;
     }
```

Mentionnons que cette implantation des algorithmes LQUEUE, LDEQUE et L2QUE possède le grand avantage qu'à la fin de ces algorithmes, tous les éléments du tableau Q[] sont nuls. Ainsi, le calcul d'une nouvelle arborescence des chemins les plus courts ne nécessite pas l'initialisation à nouveau de ces valeurs.

Nous terminons cette section en mentionnant que certains auteurs (e.g. Pallottino, 1979; Dial et al., 1979; Pape, 1980) suggèrent une autre implantation des algorithmes LDEQUE et L2QUE. Cette dernière consiste à utiliser un tableau Q1[] de dimension n+1 dont les éléments sont définis de la façon suivante :

$$Q1[i] = \begin{cases} j & \text{si i est dans la liste et j est le sommet suivant,} \\ 0 & \text{si i n'est pas dans la liste et n'y est jamais entré,} \\ -1 & \text{si i n'est pas dans la liste, mais y a déjà été,} \\ n+1 & \text{si i est le dernier de la liste.} \end{cases}$$

Ainsi, quand un sommet est retiré de la liste, nous faisons "Q[i]=-1" au lieu de "Q[i]=0". En plus, lorsque ces algorithmes sont implantés en FORTRAN, il suffit d'exécuter l'instruction

```
      IF (Q1[j]) 100, 200, 300
100   traiter le cas Q1[j]<0
 .
 .
 .
200   traiter le cas Q1[j]=0
 .
 .
 .
300   traiter le cas Q1[j]>0
```

pour traiter l'ajout du sommet j dans Q.

Finalement, il convient de mentionner que Pallottino (1979) suggère d'utiliser -k, où k correspond au $k^{ième}$ appel à la procédure de recherche des chemins le plus court, à la place de -1. Cela évite de réinitialiser

les valeurs du tableau Q1[]. Toutefois, en langage C, cette méthode pose quelques problèmes d'implantation car les éléments de Q[] sont des pointeurs (quoi faire avec -k?).

4.4 Algorithme LTHRS

La caractéristique des algorithmes LDEQUE et L2QUE est le partitionnement implicite de la liste des sommets à explorer selon qu'un sommet reçoit une amélioration pour la première fois ou non. Récemment, Glover et al. (1985a, 1985b) ont introduit l'utilisation d'une fonction Seuil pour diviser la liste des sommets candidats. À partir de cette nouvelle idée, il est possible de mettre en oeuvre plusieurs algorithmes dépendant du choix de la fonction Seuil et de la politique d'insertion des sommets dans la liste des candidats. Dans ce mémoire, nous décrivons l'implantation, en langage C, d'une variante de l'algorithme TRESHX2 (voir Glover et al., 1985b). Notons que notre implantation est fortement inspirée de l'article de Gallo et Pallottino (1988). D'ailleurs, nous référons le lecteur à ce papier pour la description de l'algorithme en pseudo-code ou FORTRAN.

Dans cet algorithme, la liste des sommets à explorer est partitionnée en deux listes distinctes NOW et NEXT. Contrairement aux algorithmes précédents, ici nous employons deux tableaux, NOW[] et NEXT[], de dimension n+1 dont les indices sont les numéros des sommets. La liste NOW est une file qui contient tous les sommets dont la valeur des étiquettes est inférieure ou égale à la variable Seuil, THRS. Les autres sommets sont placés dans la liste chaînée NEXT.

Notre implantation en langage C emploie la structure "NODE" suivante :

```
struct NODE {
            struct ARC *a;      /*tableau A[]*/
            struct NODE *p;     /*tableau P[]*/
            int d;              /*tableau D[]*/
            struct NODE *now;   /*tableau NOW*/
            struct NODE *next; /*tableau NEXT*/
           }node [n+2];
```

L'algorithme débute avec THRS valant zéro et en plaçant le sommet r dans la file NOW. La procédure d'initialisation est la suivante :

Procédure INIT (Q,r) (algorithme LTHRS)

```
struct NODE *ptr, *nl, *last;
int THRS;
ptr = (node+1); nl = (node+n+1)
while (ptr < nl)
   {
    ptr → p = r;
    ptr → d = HUGE;
    ptr → now = ptr → next = '\000';
    ptr ++;
   }
r → p = '\000'; r → d = 0;
r → now = nl;                    /*insertion du sommet r dans NOW*/
nl → now = last = r;
nl → next = nl;
THRS = 0;
```

La politique d'insertion consiste à placer les sommets dont l'étiquette est plus petite ou égale à la valeur Seuil de la file NOW, les autres étant mis dans NEXT. De plus, pour des raisons d'efficacité, Gallo et Pallottino (1988) acceptent d'introduire un sommet dans NOW, même s'il est déjà dans NEXT. C'est dans la procédure QOUT() que les copies des sommets sont éliminées.

Procédure QIN (j) (algorithme LTHRS)

```
if (j → d < THRS)
   {
    if (!j → now)            /*si j n'est pas déjà dans NOW*/
     {
      last → now = j;        /*insérer j à la fin de la liste NOW*/
      j → now = nl;
      last = j;
     }
```

```
else if (!j → next)        /*si j n'est pas déjà dans NEXT*/
 {
  j → next = n1 → next;  /*placer j dans NEXT*/
  n1 → next = j;
 }
```

Le choix à chaque étape est de prendre le premier élément de la file NOW. Quand celle-ci est vide, la valeur de la variable THRS est mise à jour et nous transférons des sommets de NEXT à NOW selon la nouvelle valeur de THRS. L'ordinagramme de la figure 4.4 montre le déroulement de la procédure QOUT(i). Puisque cette procédure est assez longue, nous ne la donnons pas ici. Rappelons que nos programmes sont disponibles sur diskette.

Dans la figure 4.4, nous remarquons que la mise à jour de la variable THRS est effectuée à l'aide de la formule suivante :

$$THRS = \begin{cases} THRS + TINC + 1 & \text{si } min < THRS + TINC + 1 \\ min + TINC & \text{sinon} \end{cases}$$

où $min = \underset{i \in NEXT}{Min} \{d_j\}$

Ainsi, l'efficacité de l'algorithme repose sur la valeur du paramètre TINC ("threshold increment"). Notons que si TINC vaut zéro, l'algorithme transfère, à chaque étape, le sommet ou les sommets de l'ensemble NEXT qui possèdent l'étiquette dont la valeur est la plus petite. Dans ce cas, cela se ramène à une implantation de la méthode de Dijkstra. D'autre part, si la valeur de TINC est plus grande ou égale à lmax (où $lmax = Max \{\ell_{ij} : (i,j) \in U\}$), à chaque étape, tous les sommets sont transférés de NEXT à NOW. Cela correspond à l'algorithme LQUEUE. Ce sont ces considérations qui nous permettent d'interpréter l'algorithme avec fonction Seuil comme une méthode hybride alliant les avantages des algorithmes par extension sélective et par ajustement progressif.

Glover et al. (1985b, algorithme THRESHX2) utilisent la formule suivante pour déterminer TINC :

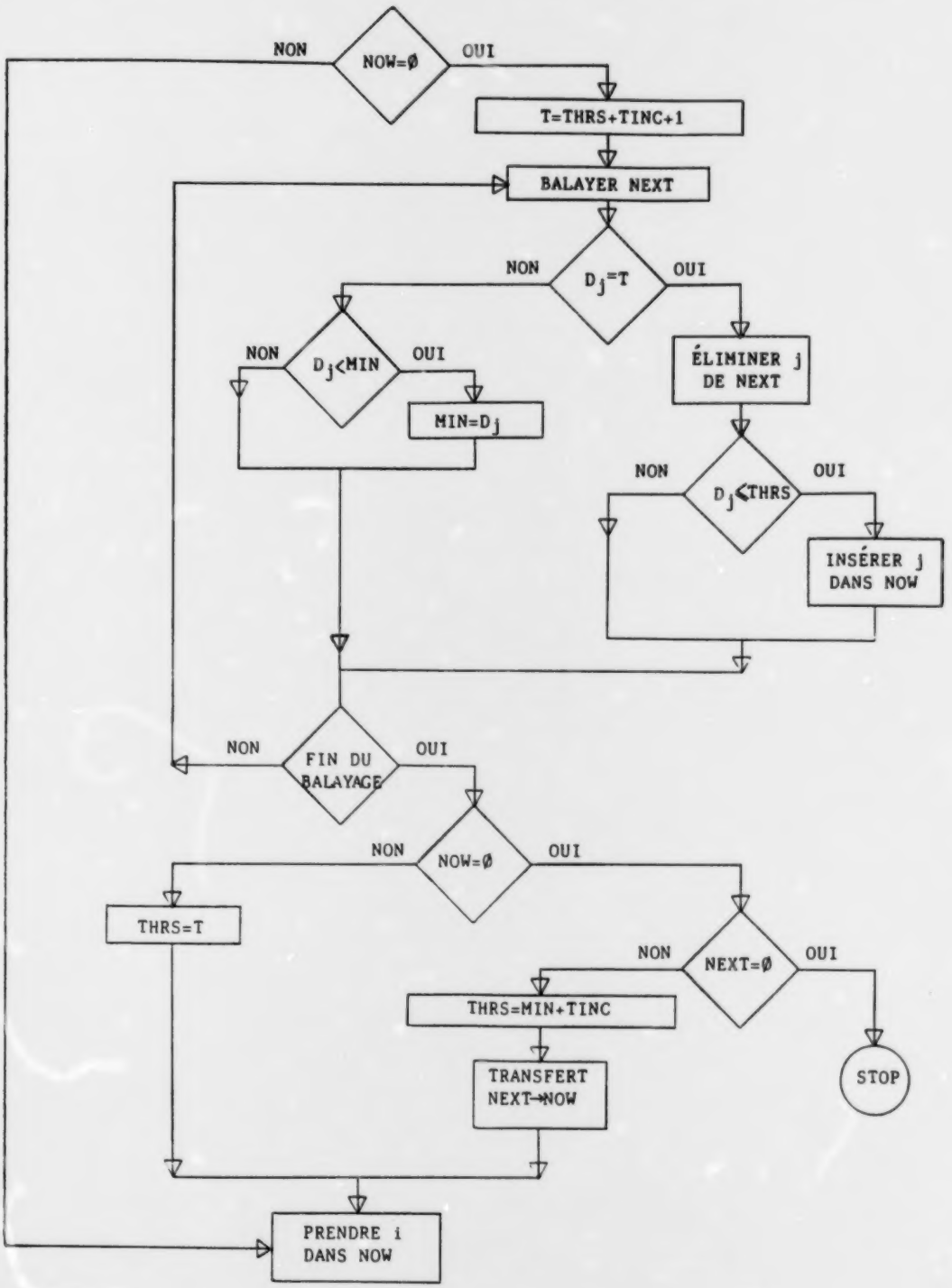


Figure 4.4 : Déroulement de la procédure QOUT

$$TINC = \begin{cases} x2 \cdot lmax & \text{si DENSE} < 7 \\ \dfrac{x2 \cdot lmax \cdot 7}{DENSE} & \text{si DENSE} > 7 \end{cases}$$

où $DENSE = Min\left\{\frac{m}{n}, 35\right\}$

$$x2 = \begin{cases} 0.25 & \text{pour les réseaux aléatoires,} \\ 1.5 & \text{pour les réseaux en grille.} \end{cases}$$

Mentionnons que ces auteurs ne justifient pas les termes (7,35,0.25,1.5) employés dans cette fonction. Cependant, il est fort probable que ces nombres aient été calculés à l'aide de nombreuses expérimentations numériques. Enfin, nous référons le lecteur à Divoky (1987) pour la mise au point et l'étude d'autres fonctions TINC et THRS.

4.5 Algorithme SDKSTR

La mise en oeuvre la plus simple d'une file de priorité réside dans l'utilisation d'une liste simplement chaînée qui est balayée à chaque itération afin de trouver l'élément possédant l'étiquette dont la valeur est la plus faible. À la section 4.3, nous avons vu que l'implantation la plus efficiente d'une liste simplement chaînée repose sur l'emploi d'un tableau Q[] de dimension n+1 dont les éléments sont définis de la façon suivante :

$$Q[i] = \begin{cases} j & \text{si i est dans la liste et j est le suivant,} \\ 0 & \text{si i n'est pas dans la liste,} \\ n+1 & \text{si i est le dernier élément de la liste.} \end{cases}$$

Aussi, le numéro du premier élément de la liste se trouve à la case n+1 de ce tableau afin de rendre la liste circulaire. En langage C, cela se traduit par l'utilisation de la structure "NODE" que voici :

```
struct NODE {
            struct ARC *a;      /*tableau A[]*/
            struct NODE *p;     /*tableau P[]*/
            int d;              /*tableau D[]*/
            struct NODE *q;     /*tableau Q[]*/
           }node [n+2];
```

Avec cette structure de données, la procédure d'initialisation INIT() prend la forme suivante :

Procédure INIT (Q,r) (algorithme SDKSTR)

```
struct NODE *ptr, *n1;
int min;                /*le rôle de min est décrit plus loin*/
ptr = (node+1); n1 = (node+n+1);
while (ptr < n1)
   {
    ptr → p = r;        /*P[i]=r*/
    ptr → d = HUGE;     /*D[i]=HUGE*/
    ptr → q = '\000';   /*Q[i]=0*/
    ptr ++;
   }
r → p = '\000'; r → d = 0;
r → q = n1;
n1 → q = r;
```

Étant donné que l'ordre dans lequel sont placés les sommets n'a pas d'importance (car toute la liste doit être balayée à chaque itération), nous avons choisi d'ajouter les nouveaux sommets au début de la liste. Cela donne la procédure QIN() suivante :

Procédure QIN (j) (algorithme SDKSTR)

```
if (!j → q)
   {                    /*insertion au début*/
    j → q = n1 → q;     /*Q[j]=Q[n+1]*/
    n1 → q = j;         /*Q[n+1]=j*/
   {
```

Finalement, nous retirons le sommet qui possède l'étiquette dont la valeur est la plus faible avec la procédure QOUT(i) qui nécessite le parcours de toute la liste.

Procédure QOUT (i) (algorithme SDKSTR)

```
min = HUGE;              /*à la fin de cette séquence, min représente
                           la valeur de l'étiquette la plus faible*/
i = n1; ptr = i;
while (i → q != n1)                  /*tant que la liste est non vide*/
   {
    if (i → q → d < min)             /*if D[Q[i]] < min*/
       {
        ptr = i; min = i → q → d; /*min=D[Q[i]]*/
       }
    i = i → q;
   }
i = ptr → q; ptr → q = i → q; i → q = '\000';
```

Comme précédemment, les éléments du tableau Q[] sont nuls à la fin de l'algorithme. Ainsi, il est suffisant de les initialiser une seule fois.

4.6 Algorithme SHEAP

En pratique, nous disons que les sommets, numérotés 1,2,...,n, qui appartiennent à une file de priorité forment un tas ("binary heap") si leur numéro sont dans un vecteur HP[] de dimension n, tel que

$$D[HP[k]] \geq D[HP[\lfloor \frac{k}{2} \rfloor]] \quad \text{pour tout } k=2,3,\ldots,nhp$$

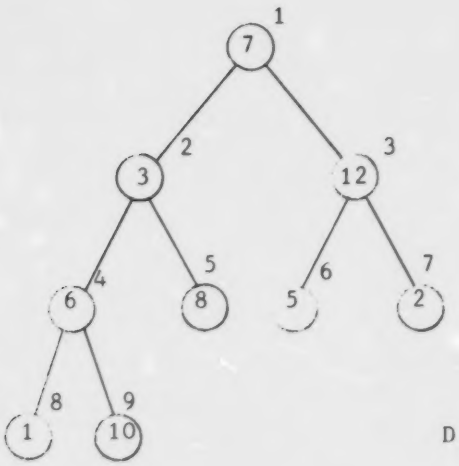
où $\lfloor \frac{k}{2} \rfloor$ correspond à la partie entière de $\frac{k}{2}$, et nhp est le nombre de sommets dans la file de priorité ($nhp \leq n$).

HP[k] est donc le numéro du sommet placé en position k. Dans le but de rendre efficace les opérations de mise à jour de la file de priorité, nous employons aussi un tableau Q[] de dimension n dont les indices sont les numéros des sommets et dont les éléments sont définis de la façon suivante:

$$Q[i] = \begin{cases} k & \text{si } HP[k]=i \\ 0 & \text{si le sommet } i \text{ n'est pas dans la file de priorité.} \end{cases}$$

Dans la littérature scientifique, le tableau Q[] est appelé le dictionnaire de la file de priorité (voir Fox, 1978).

Cette représentation d'un tas dans la mémoire centrale de l'ordinateur est illustrée à la figure 4.5. Les noeuds de l'arbre binaire sont indicés 1,2,...,nhp, de haut en bas et de gauche à droite ("breadth first order"; voir Tarjan, 1983). Ainsi, les successeurs d'un noeud k de l'arbre binaire sont les noeuds 2k et 2k+1. De même, le prédécesseur d'un noeud k est le noeud $\left\lfloor \frac{k}{2} \right\rfloor$. Mentionnons qu'en langage C, les multiplications et les divisions par des multiples de 2 sont calculées plus rapidement par des décalages à gauche ou à droite du nombre de bits approprié.



	1	2	3	4	5	6	7	8	9	10	11	12
D	21	9	6	*	7	8	4	15	*	14	*	5
Q	8	7	2	0	6	4	1	5	0	9	0	3
HP	7	3	12	6	8	5	2	1	10			

* = INFINI

Figure 4.5 : Représentation d'un tas

En langage C, il n'est pas permis de multiplier ni diviser un pointeur par une valeur numérique. Pour cette raison, il n'est pas utile que les éléments du tableau Q[] soient des pointeurs. Donc nous implantons l'algorithme SHEAP à l'aide de la structure "NODE" ci-bas :

```
struct NODE {
            struct ARC *a;      /*tableau A[]*/
            struct NODE *p;     /*tableau P[]*/
            int d;              /*tableau D[]*/
            int q;              /*tableau Q[]*/
           }node [n+2];
```

et d'un tableau hp[] dont les éléments sont de pointeurs sur les sommets :

```
struct NODE *hp[n+1];
```

Au début de l'algorithme, le sommet r est le seul élément placé dans le tas. Voici la procédure d'initialisation en langage C :

Procédure INIT (Q,r) (algorithme SHEAP)

```
struct NODE *ptr, *n1
int nhp;
ptr = (node+1); n1 = (node+n+1)
while (ptr < n1)
   {
    ptr → p = r;        /*P[i]=r*/
    ptr → d = HUGE;     /*D[i]=HUGE*/
    ptr → q = 0;        /*Q[i]=0*/
    ptr ++;
   }
r → p = '\000'; r → d = 0;
nhp = 1;
hp[1] = r; r → q = 1;  /*Q[r]=1*/
```

Nous constatons que les procédures pour l'insertion et la suppression des sommets dans un tas ont été largement décrites dans la littérature

scientifique. Ainsi, nous référons le lecteur aux ouvrages de Williams (1964), Standish (1980), Sedgewick (1983) et Tarjan (1983) pour l'étude de celles-ci. Pour notre part, nous terminons cette section en mentionnant que les détails en langage C des fonctions QIN() et QOUT() sont disponibles sur diskette. Nous justifions cette décision de ne pas les présenter dans cette section en faisant remarquer que notre implantation est semblable à celle décrite en pseudo-code par Gallo et Pallottino (1988).

4.7 Tri par compartiments : algorithmes SDIAL et SRHEAP

Dans cette section, nous présentons les deux algorithmes SDIAL et SRHEAP car leur mise en oeuvre repose sur l'emploi de compartiments dans lesquels les sommets de la file de priorité sont placés. La différence entre les deux algorithmes réside dans le nombre de compartiments utilisés et dans les règles d'insertion et de suppression des sommets.

4.7.1 Tri par compartiments

Denardo et Fox (1979) ont analysé trois façons de conserver les sommets dans un compartiment : les listes doublement chaînées, les listes simplement chaînées et l'utilisation des bits ("bit-pack"). Soulignons que l'ordre dans lequel les sommets apparaissent dans les compartiments n'a pas d'importance. Il n'est pas nécessaire de les trier selon la valeur de leur étiquette, par exemple. Ils ont montré une implantation efficiente des listes doublement chaînées et ils sont parvenus à la conclusion que cette stratégie est celle qui requiert le moins d'accès à la mémoire. Cette implantation consiste à utiliser les tableaux Q[], UP[] et DOWN[], dont les dimensions respectives sont K, n+1 et n+1. Dans cette section, K désigne le nombre de compartiments induit par la fonction d'adressage, et pntr est un compartiment. La figure 4.6 illustre cette structure de données. La case Q[pntr] est l'indice du premier sommet dans la liste des sommets du compartiment pntr. Par convention, nous posons Q[pntr]=n+1 quand le compartiment

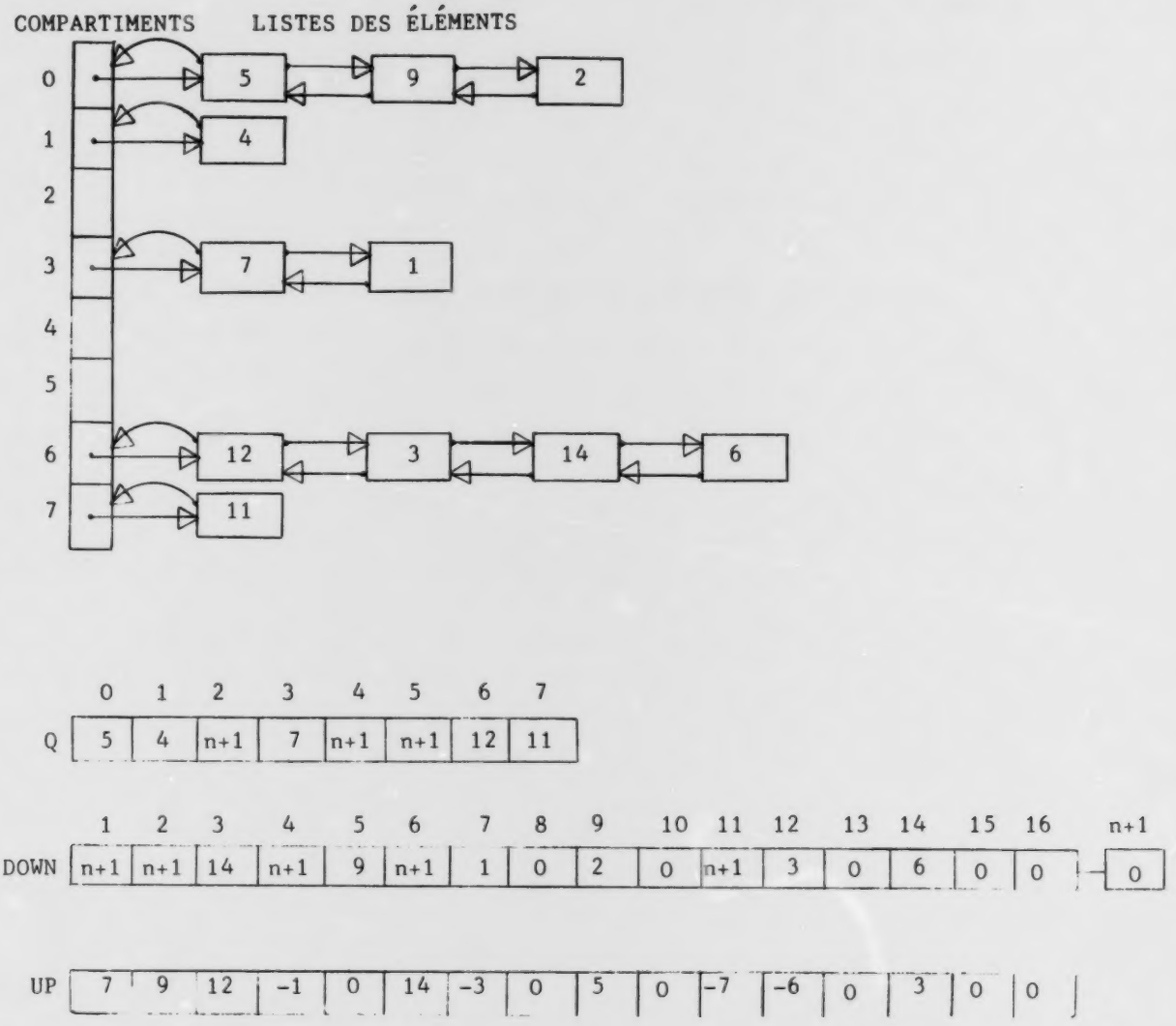


Figure 4.6 : Schéma d'une structure de compartiments implantée avec des listes doublement chaînées

pntr est vide. Dans cette section, le rôle du tableau Q[] est donc de représenter la liste des compartiments. Les tableaux UP[] et DOWN[] servent à chaîner les sommets d'un compartiment tel que

$$\text{DOWN}[i] = \begin{cases} j & \text{si } j \text{ est le prochain sommet de la liste,} \\ 0 & \text{si } i \text{ n'est pas dans la file de priorité,} \\ n+1 & \text{si } i \text{ est le dernier sommet d'un compartiment.} \end{cases}$$

$$\text{UP}[j] = \begin{cases} i & \text{si } i \text{ précède } j \text{ dans la liste d'un compartiment,} \\ 0 & \text{si } j \text{ n'est pas dans la file de priorité,} \\ -\text{pntr} & \text{si } j \text{ est le premier sommet du compartiment pntr.} \end{cases}$$

Par conséquent, les opérations

- insert (j,pntr) insertion du sommet j dans le compartiment pntr,
- delete (j) suppression du sommet j de la file de priorité,
- delete first (pntr) suppression du premier sommet du compartiment pntr, la procédure retourne le sommet éliminé.

requièrent O(1) calculs. Mentionnons aussi que cette structure de données permet d'exécuter ces opérations sans être obligé de tester les cas spéciaux (e.g. liste d'un seul élément, dernier élément de la liste) rencontrés dans le vieux problème "PASCAL" concernant la réalisation d'insertions et de suppressions à n'importe quelle position d'une liste. Le lecteur peut trouver le listing, en pseudo-code, de ces procédures dans Gallo et Pallottino (1988).

Malheureusement, en langage C, il est impossible d'implanter cette structure de données avec des pointeurs, à cause de la signification double des éléments du tableau UP[]. Les éléments positifs indiquent une position dans le tableau UP[] tandis que les éléments négatifs font référence au tableau Q[].

Dans ce cas, une structure de données alternative, analysée par Denardo et Fox (1979), consiste à employer des listes simplement chaînées et un dictionnaire. Lorsque nous avons à déplacer un sommet d'un compartiment à un autre, nous devons d'abord le retrancher du compartiment où il se trouve. La procédure consiste à remplacer le vide créé par la suppression d'un sommet, appelons-le j, par le premier sommet du compartiment auquel j appartient. L'usage du dictionnaire revient donc à utiliser une méthode d'adressage indirecte. Ainsi, nous constatons que nous pouvons enlever un sommet d'un compartiment sans avoir à effectuer les tests pour les cas spéciaux ni à parcourir la liste pour trouver l'élément précédent.

L'efficacité de la méthode dépend alors de la rapidité avec laquelle nous accédons aux éléments. Denardo et Fox (1979) ont souligné que cette structure de données semble mieux adaptée au langage assembleur qu'au FORTRAN. Puisque le langage C peut manipuler efficacement des adresses (dans ce sens, le C s'apparente au langage assembleur), nous avons implanté cette option.

Les informations concernant les noeuds sont emmagasinées dans le tableau **node[]** défini ci-bas :

```
struct NODE {
             struct ARC *a;
             struct NODE *p;
             int d;
             struct DCTNRY *beta;
            }node [n+2];
```

Le champ **beta** contient l'adresse, dans le dictionnaire, où se trouve ce noeud.

La liste des compartiments est contenue dans le tableau **bucket[]** suivant :

```
struct BUCKET {
               struct DCTNRY *head;
              } BUCKET [k];
```

Le champ **head** représente le premier élément de ce compartiment. Notons que les éléments des compartiments sont des adresses du dictionnaire.

Enfin, le tableau **dctnry[]**, jouant le rôle du dictionnaire, est défini de la façon suivante :

```
struct DCTNRY {
               struct DCTNRY *down;
               struct BUCKET *assgn;
               struct NODE *alpha;
              } dctnry [n+2];
```

Le champ **down** contient l'adresse du prochain élément dans le compartiment. Il maintient les listes chaînées. Le champ **assgn** représente l'adresse du compartiment où cet élément se trouve. Cette donnée est nécessaire lorsque nous avons à déplacer un noeud d'un compartiment à un autre. Le dernier champ, **alpha**, est un pointeur qui donne accès aux informations concernant les noeuds.

Voici maintenant le détail, en langage C, des trois opérations **insert (j,pntr)**, **delete (j)** et **delete first (pntr)** qui utilisent la structure de compartiments implantée avec des listes simplement chaînées et un dictionnaire :

Procédure insert (j,pntr)

Afin d'insérer un sommet j dans le compartiment pntr, nous commençons par identifier sa position dans le dictionnaire. Ensuite, il suffit de placer cette adresse à la tête du compartiment. Rappelons que l'ordre dans

lequel sont ajoutés les sommets n'est pas important. Voici la définition des variables et la séquence :

```
struct NODE *j;
struct BUCKET *pntr;
struct DCTNRY *newhead;
newhead = j → beta;          /*position de j dans le dictionnaire*/
newhead → down = pntr → head;
pntr → head = newhead;       /*insertion au début*/
newhead → assgn = pntr;
```

Procédure delete (j)

La procédure qui permet de retrancher un sommet de la file de priorité est celle qui demande le plus d'accès à la mémoire. Elle consiste à remplacer le vide créé par la suppression du sommet j par le premier sommet du compartiment auquel j appartient, en interchangeant des pointeurs du dictionnaire. Voici la séquence, en langage C, qui accomplit cette tâche, les commentaires indiquant les principales étapes :

```
struct NODE *j, *dummy;
struct BUCKET *pntr;
struct DCTNRY *pj, *oldhead;
/*initialisation*/
pj = j → beta;               /*position de j dans le dictionnaire*/
pntr = pj → assgn;           /*compartiment auquel j appartient*/
oldhead = pntr → head;       /*premier élément de ce compartiment*/
/*combler le vide laissé par la suppression de j*/
dummy = oldhead → alpha;     /*premier noeud du compartiment*/
oldhead → alpha = j;
pj → alpha = dummy;          /*interchanger les pointeurs du*/
dummy → beta = pj;           /*dictionnaire*/
j → beta = oldhead;
/*déplacer la tête du compartiment*/
pntr → head = oldhead → down;
pj → assgn = '\000'; oldhead → down = '\000';
```

Procédure delete first (pntr)

Cette procédure retourne le premier sommet du compartiment **pntr** et elle procède de la façon suivante :

```
struct NODE *i;
struct DCTNRY *oldhead;
oldhead = pntr → head;
pntr → head = oldhead → down;
oldhead → assgn = ' 000';
oldhead → down = ' 000';
i = oldhead → alpha;
return (i);
```

Les deux prochaines sous-sections sont consacrées à l'étude de la mise en œuvre, en langage C, des algorithmes SDIAL et SRHEAP qui s'appuient sur les procédures que nous venons de présenter.

4.7.2 Algorithme SDIAL

La fonction d'adressage dispersée utilisée par Dial (voir chapitre 3) fait usage de K=lmax+1 compartiments, numérotés 0,1,...,lmax. Ceux-ci sont balayés d'une manière circulaire par le pointeur **pntr.** Le premier compartiment dont le contenu est non vide contient tous les sommets dont la valeur de l'étiquette est la plus faible. À chaque fois que l'étiquette d'un sommet est diminuée, le sommet est transféré dans un nouveau compartiment. Initialement, le sommet r est placé dans le compartiment 0. La procédure INIT() est la suivante :

Procédure INIT (Q,r) (algorithme SDIAL)

```
struct NODE *ptr, *nl
struct BUCKET *first, *last, *pntr; /*premier et dernier compartiment*/
struct DCTNRY *pj;
int cntr;                           /*nombre de sommets dans la file
                                      de priorité*/
```

```
ptr = (node+1); n1 = (node+n+1)
pj = (dctnry+1);
while (ptr < n1)
   {
    ptr → p = r;
    ptr → d = HUGE;
    ptr → q = '\000';
    ptr → beta = pj;              /*initialisation du dictionnaire*/
    pj → alpha = ptr;
    pj → down = '\000';
    pj → assgn = '\000';
    ptr ++; pj ++;
   }
r → p = '\000'; r → d = 0;

/*initialisation des compartiments*/
pntr = first = bucket; last = bucket + lmax;
while (pntr <= last)
   {
    pntr → head = '\000';
    pntr ++;
   }
insert (r,first);                 /*insertion du sommet r dans la
                                    file de priorité*/
cntr = 1;
pntr = first;                     /*le rôle de pntr est décrit
                                    ci-après*/
```

Mentionnons qu'il n'est pas nécessaire de réinitialiser les compartiments, ni le dictionnaire entre chaque appel à l'algorithme SDIAL. En effet, les champs **head**, **down** et **assgn** sont tous nuls à la fin du calcul des chemins les plus courts.

La procédure QOUT() consiste à balayer, d'une façon circulaire, les compartiments afin d'y trouver le premier non vide. Pour ce faire, nous utilisons la variable **pntr** qui est un pointeur sur le premier compartiment non vide. Voici la procédure QOUT() :

Procédure QOUT (i) (algorithme SDIAL)

```
while (!pntr → head)
   {
    if (pntr == last) pntr = first;
    else pntr ++;
   }
i → delete first (pntr);
cntr --;
```

La variable **cntr** vaut le nombre de sommets dans la file de priorité. Le critère d'arrêt de l'algorithme est donc "**cntr** == 0".

Quand l'étiquette d'un sommet est amélioré pour la première fois, ce sommet est inséré dans la file de priorité. Autrement, un sommet dont l'étiquette est diminuée pour au moins une seconde fois doit d'abord être retiré du compartiment où il se trouve pour ensuite être réinséré au bon endroit. Cela est accompli par la procédure QIN() que voici :

Procédure QIN (j) (algorithme SDIAL)

```
struct BUCKET *x;       /*variable tampon*/
if (!j → alpha → assgn)
   {
    cntr ++;            /*j n'est pas dans la file de priorité*/
   }
else
   {                    /*j est dans un compartiment*/
    delete (j);         /*nous devons l'ôter*/
   }

x = pntr + k → lngth;  /*calcul de l'adresse du nouveau compartiment*/
if (x > last) x = first + (x-last-1);
insert (j,x);
```

4.7.3 Algorithme SRHEAP

La particularité de l'algorithme SRHEAP est que la largeur des intervalles d'admission des sommets dans les compartiments n'est pas la même pour tous les compartiments. De plus, ces intervalles sont modifiés par l'algorithme. Ainsi, nous devons garder explicitement les bornes inférieures et supérieures associées à chacun des compartiment. Pour ce faire, nous ajoutons les champs **l** ("lower") et **u** ("upper") à la structure "BUCKET" présentée précédemment. Voici la nouvelle structure :

```
struct BUCKET {
                struct DCTNRY *head;
                int l, u;
               }bucket [K];
```

L'algorithme fait usage de K+1, où $K = 1 + \lceil \log lmax \rceil$ compartiments, numérotés 0,1,...,K. Les valeurs initiales des bornes sont :

$$\text{bucket [pntr] . l} = \begin{cases} 0 & \text{si pntr} = 0 \\ 1 & \text{si pntr} = 1 \\ 2^{\text{pntr}-1} & \text{si pntr} = 2,3,\ldots,K \end{cases}$$

$$\text{bucket [pntr] . u} = \begin{cases} 0 & \text{si pntr} = 0 \\ 1 & \text{si pntr} = 1 \\ 2^{\text{pntr}}-1 & \text{si pntr} = 2,3,\ldots,K \end{cases}$$

Comme précédemment, les pointeurs **first** et **last** indiquent le premier et le dernier compartiment tandis que le pointeur **pntr** sert au balayage des compartiments. La variable **cntr** vaut le nombre de sommets dans la file de priorité et le pointeur **pj** est utilisé pour accéder aux adresses du dictionnaire. La procédure d'initialisation est la suivante :

Procédure INIT (Q,r) (algorithme SHREAP)

```
struct NODE *ptr, *nl;
struct BUCKET *first, *last, *pntr;
struct DCTNRY *pj;
int cntr, p;

ptr = (node+1); nl = (node+n+1)
pj = (dctnry+1);
while (ptr < nl)
   {
    ptr → p = r;
    ptr → d = HUGE;
    ptr → q = '\000';
    ptr → beta = pj;
    pj → alpha = ptr;
    pj → down = '\000';
    pj → assgn = '\000';
    ptr ++; pj ++;
   }
r → p = '\000'; r → d = 0;

/*initialisation des compartiments*/
pntr = first = bucket; last = bucket + ⌈log lmax⌉ +1; p=1;
while (pntr <= last)
   {
    pntr → head = '\000';
    pntr → l = p; p=p << 1;      /*p=p+2*/
    pntr → u = p-1
    pntr ++;
   }
insert (r,first)
cntr = 1;
```

Mentionnons que Ahuja et al. (1988) emploient un compartiment supplémentaire, numéroté K+1, dans lequel sont placés tous les sommets qui n'ont pas reçu une amélioration pour la premiere fois (i.e. la valeur

de leur étiquette est HUGE). Cependant, nous croyons que cela provoque un travail inutile au moment de l'ajout ou de la réinsertion des sommets dans la file de priorité. Ainsi, puisque ce compartiment renferme n-1 sommets au début de l'algorithme, cela implique n-1 appels à la procédure **delete()**. Or, à la section 4.4.1, nous avons constaté que celle-ci nécessite plusieurs accès à la mémoire. À la place, nous testons si le sommet qui reçoit une amélioration doit être mis pour la premiere fois dans la file de priorité ou s'il doit être réinséré dans un nouveau compartiment. Cela est accompli par la procédure QIN() suivante :

Procédure QIN (j) (algorithme SHREAP)

```
struct BUCKET *x;
x = j + alpha + assgn          /*compartiment où se trouve le
                                 sommet j*/
if (!x)
   {                           /*le sommet j n'est pas dans la file
                                 de priorité*/
    x = last; cntr ++;
    while (dj < x + 1) x --;
    insert (j,x);
   }
else if (dj < x + 1)           /*doit-on réinsérer le sommet j ?*/
   {
    delete (j); x --;
    while (dj < x + 1) x --;   /*réinsertion de j*/
    insert (j,x);
   }
```

La recherche du sommet ayant l'étiquette dont la valeur est la plus faible consiste à chercher le premier compartiment non vide, à l'aide du pointeur **pntr**, en commençant par le premier. Quand la fouille s'arrête sur le compartiment numéro 0 (**first**) où 1 (**first** + 1), la procédure QIN() retourne le premier sommet de ce compartiment. La taille des intervalles de ces deux compartiments étant un, tous les sommets qui s'y trou-

vent sont des sommets d'étiquette minimum. Autrement, quand **pntr** est plus grand que **first** + 1, tous les sommets de ce compartiment sont réaffectés (redistribués) dans les précédents en recalculant les bornes des intervalles d'admission.

Ahuja et al. (1988) suggèrent la procédure suivante pour mettre à jour les intervalles : (voir chapitre 3)

```
first → l = first → u = min; p=1;
if (pntr == last)
   {
    for (x = first+1; x <  last; x ++)
       {
        x → l = min + p; p >> 1;
        x → u = min + p-1;
       }
   }
else
   {
    for (x = first+1; x <  pntr; x ++)
       {
        x → l = min + p; p >> 1;
        x → u = minimum (min + p-1, pntr → u);
       }
   }
```

Dans cette procédure, **min** représente la valeur de l'étiquette la plus faible des sommets renfermés dans le compartiment pntr. Cette valeur est obtenue par la séquence suivante :

```
pj = pntr → head; min = HUGE:
while (pj)
   {
    dj = pj → alpha → d;
    if (dj < min) min = dj;
    pj = pj → up;
   }
```

Remarquons qu'avec cette façon de calculer les nouvelles bornes, tous les compartiments inactifs (voir chapitre 3) sont caractérisés par le fait que la valeur de leur borne inférieure est plus grande que la valeur de leur borne supérieure. Toutefois, cela implique un travail supplémentaire pour la réinsertion des sommets. En effet, nous devons exécuter l'instruction suivante afin de déterminer le compartiment où placer un sommet :

```
while (!(dj < x → u  && dj > x → l)) x --;
```

où x est un pointeur sur les compartiments,
dj est la valeur de l'étiquette du sommet à insérer.

Nous proposons de fixer la valeur du champ **l** à HUGE quand un compartiment devient inactif. Cela a pour effet d'éliminer une comparaison et il suffit d'exécuter l'instruction

```
while (dj < x → l) x --;
```

pour localiser le nouveau compartiment.

Voici la séquence que nous employons pour mettre à jour les intervalles :

```
first → l = first → u = min; p=1;
if (pntr == last)
   {
    for (x = first+1; x <  last; x ++)
       {
        x → l = min + p; p >> 1;
        x → u = min + p-1;
       }
   }
else
   {
    x = first+1; upntr = pntr → u;
    while (x <  pntr)
```

```
        {
         x + l = p + min; p = p << 1;
         x + u = p + min-1;
         if (x + u > upntr)
            {
             x + u = upntr;
             if (x + l > upntr) x + l = HUGE;
             x ++; /*tous les autres compartiments sont inactifs*/
             while (x <= pntr) {x + l = HUGE; x ++;}
             break;
            }
         x ++;
        }
    }
```

Mentionnons qu'avec cette structure de données, le nombre de compartiments nécessaires est très peu élevé. Remarquons que le plus grand entier est 2147483647 (2^{31}-1) sur un ordinateur quand la taille des entiers occupe 32 bits (SUN 3/180, IBM PS/2, carte DEFINICON DSI-32, par exemple). Donc K est borné par 33. De plus, en pratique, lmax est beaucoup plus petit que cela (lmax = 10 000 alors K=16). Par conséquent, l'initialisation des compartiments et leur balayage sont des opérations peu coûteuses en temps CPU, contrairement à l'algorithme SDIAL.

4.8 Résultats numériques

Dans cette section, nous décrivons les expérimentations numériques que nous avons menées et nous analysons les résultats obtenus. Nous discutons aussi des impacts de la programmation en langage C sur l'efficacité des algorithmes.

Rappelons qu'un des buts de ce mémoire de maîtrise consiste à comparer les algorithmes de recherche des chemins minimaux quand leur

implantation prend avantage des caractéristiques du langage C. Toutefois, nous ne pouvons pas comparer les temps d'exécution entre des programmes écrits en FORTRAN et en C. En effet, les deux langages ne possèdent pas les mêmes instructions pour préciser l'ordre dans lequel se font les calculs. Le FORTRAN ne dispose pas de l'instruction **while**, par exemple. À la place, il fait usage de nombreux **goto**. De plus, il est reconnu que les programmes en C sont plus rapides. Nous avons donc effectué, en langage C, une seconde implantation des algorithmes LQUEUE, LDEQUE, L2QUE, LTHRS, SDKSTR, SHEAP, SDIAL et SRHEAP en employant des tableaux et des faux-pointeurs, le C le permettant. Les listings de ces programmes sont aussi disponibles sur disquette. Soulignons que ces programmes sont semblables aux procédures en pseudo-code de Gallo et Pallottino (1988). De plus, ils ont la même structure que les programmes que nous avons décrits dans ce chapitre, sauf pour SDIAL et SRHEAP pour lesquels nous avons utilisé une structure de données alternative.

Dans cette section, les implantations qui prennent avantage des caractéristiques du C sont désignées par le terme "POINTEUR", les autres étant spécifiées par le terme "TABLEAU".

Nous avons écrit notre propre générateur de réseaux en nous inspirant du travail de Latour (1987). Le programme est écrit en C et il est disponible sur disquette. De plus, les tailles des réseaux que nous avons considérés sont celles employées par Gallo et Pallottino (1988). Les longueurs des arcs proviennent d'une distribution uniforme sur les entiers 1-100 et 1-10000. Enfin, nous avons considéré trois types de structures de réseaux :

- **Réseau complet**

  Un réseau de n sommets où chacun des sommets est joint à tous les autres. Dans ce réseau, le nombre d'arcs est $m=n(n+1)$.

- **Réseau aléatoire**

  Un réseau connexe comportant n sommets et m arcs, sans arc parallèle ni boucle.

- **Réseau grille**

  Un réseau dont les sommets et les arcs forment une grille de p rangées et q colonnes. Il y a n=pq sommets et le nombre d'arcs est m=4pq-2p-2q et où chaque sommet est joint par un arc à ses voisins immédiats.

Nous avons effectué nos expérimentations numériques à l'aide de l'ordinateur SUN 3/180 du Centre de recherche sur les transports. Le tableau 4.1 résume les caractéristiques de cette machine. Les programmes ont été exécutés la nuit, là où la demande d'activité sur le système est la plus faible. Le temps CPU est mesuré en 1/60 seconde. Dans les tableaux et les graphiques, nous rapportons le temps CPU total pris pour la résolution de dix arborescences, pour un même réseau généré, dont les origines ont été choisies au hasard. Comme il se doit, les entrées/sorties et les appels de fonctions sont exclus dans le calcul du temps total.

Tableau 4.1 : Caractéristiques de l'ordinateur SUN 3/180 utilisé au Centre de recherche sur les transports

SUN 3/180 (station crts) Centre de recherche sur les transports	
Microprocesseur (CPU), 32 bits	MC 68020
Fréquence de l'horloge	16 Mhz
Coprocesseur mathématique	MC 68887
Performance	2 MIPS
Mémoire	16 MO
Temps d'un cycle	270 ms

4.8.1 Réseau complet

Étant donné la quantité énorme d'espace requise pour les graphes complets, nous nous sommes restreints à l'étude des réseaux dont la taille est petite, soit n = 25, 75, 125 et 175.

Mentionnons que Glover et al. (1985a, 1985b) ne suggèrent pas de valeur pour le paramètre **x2** (voir section 4.4) pour les réseaux complets. Cependant, Gallo et Pallottino (1988) ont testé différentes valeurs et ils ont employé x2=0,25. Nous avons donc pris cette valeur.

Selon les résultats obtenus par Gallo et Pallottino (1988), l'algorithme le plus rapide pour les réseaux complets est une implantation de la méthode de Dijkstra avec un tas (SHEAP). Remarquons que les algorithmes utilisant des listes, sauf la procédure avec fonction Seuil, sont les plus lents. Cela est causé par le nombre très grand de chemins alternatifs qui existent dans ces réseaux. Ainsi, un très grand nombre de corrections doivent être effectuées avant l'obtention d'un chemin optimal. Notons aussi que le nombre de chemins alternatifs augmentent avec la valeur de **lmax** quand les longueurs des arcs sont des nombres entiers. en effet, quand **lmax** est grand, la probabilité que deux chemins aient la même longueur est faible (seulement dans le cas où les longueurs des arcs sont des nombres entiers). Mentionnons que leurs résultats montrent qu'en FORTRAN, la procédure de Dial est beaucoup affectée par la valeur de **lmax** et que l'algorithme LTHRS se comporte de la même façon que SHEAP quand **lmax** vaut 100.

Les résultats obtenus par nos implantations, en C, usant des tableaux et des faux-pointeurs, sont montrés dans le tableau 4.2. Ces résultats révèlent que les algorithmes LQUEUE, LDEQUE et L2QUE affichent des mauvaises performances pour les réseaux complets. Nous constatons que l'algorithme SRHEAP est plus rapide que SHEAP pour les grandes valeurs de

n. Quand **lmax** vaut 100, l'algorithme le plus rapide est LTHRS tandis que SHEAP est le plus lent parmi les algorithmes par ajustements sélectifs. Enfin, SDIAL est la procédure la plus efficiente pour les grandes valeurs de n avec **lmax** valant 10 000. Ceci est une surprise, car dans la littérature scientifique (Gallo et Pallottino, 1988; Dial et al., 1979), l'algorithme de Dial est reconnu pour ses piètres performances pour les grandes valeurs de **lmax**. Nous expliquons nos résultats en mentionnant qu'en C, même si nous utilisons des tableaux et des curseurs, les accès aux tableaux sont plus rapides qu'en FORTRAN car un renvoi à un tableau est toujours transformé par le compilateur en la création d'un pointeur. De plus, l'incrémentation de la variable, disons pntr, servant au balayage des compartiments est effectuée par l'instruction unaire, pntr ++, ceci étant plus rapide que l'instruction pntr = pntr+1. Finalement, l'usage d'un registre contribue aussi à diminuer le temps d'accès à la variable pntr.

Tableau 4.2 : Réseau complet, implantations traditionnelles

ALGORITHMES	n=25	n=75	n=125	n=175
	lmax = 100			
L2QUE	10	79	274	497
LDEQUE	9	86	297	517
LQUEUE	7	64	290	378
LTHRS	**6**	40	**108**	**204**
SDIAL	6	42	120	228
SDKSTR	6	53	141	267
SHEAP	6	48	132	281
SRHEAP	9	**37**	123	228
	lmax = 10 000			
L2QUE	12	100	378	748
LDEQUE	11	105	389	779
LQUEUE	11	78	251	480
LTHRS	**6**	**43**	134	270
SDIAL	14	51	**125**	**233**
SDKSTR	6	51	154	301
SHEAP	6	53	147	273
SRHEAP	10	56	126	245

Le tableau 4.3 montre les temps d'exécution des algorithmes quand leur implantation prend avantage des caractéristiques du C. Dans tous les cas, sauf un, la procédure la plus rapide est LTHRS, l'autre étant SRHEAP.

Tableau 4.3 : Réseau complet, implantations avec des pointeurs

ALGORITHMES	n=25	n=75	n=125	n=175
	lmax = 100			
L2QUE	7	68	216	391
LDEQUE	7	63	223	408
LQUEUE	7	49	149	293
LTHRS	**5**	**32**	**89**	**160**
SDIAL	5	36	92	182
SDKSTR	5	39	110	221
SHEAP	6	37	101	195
SRHEAP	7	39	89	166
	lmax = 10 000			
L2QUE	8	73	292	570
LDEQUE	8	82	313	590
LQUEUE	7	63	192	396
LTHRS	**3**	**29**	**86**	215
SDIAL	12	34	93	196
SDKSTR	4	48	115	219
SHEAP	6	45	106	188
SRHEAP	12	43	105	**186**

Nous constatons que les programmes usant des pointeurs sont toujours plus rapides que les autres, usant des tableaux et des faux-pointeurs. Nous avons illustré cette situation, pour le cas du réseau complet de 175 noeuds dont la longueur du plus long arc est 10 000, à la figure 4.7. La figure 4.8 montre le niveau d'amélioration obtenu, mesuré en pourcentage, pour chacun des algorithmes.

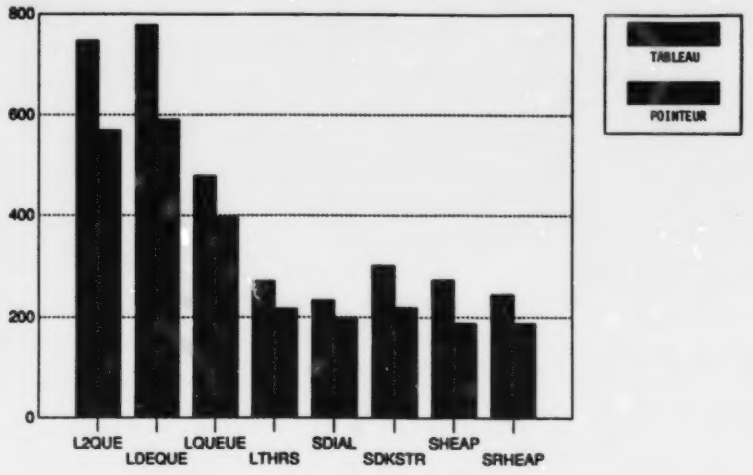


Figure 4.7 : Réseau complet : lmax = 10 000, n=175

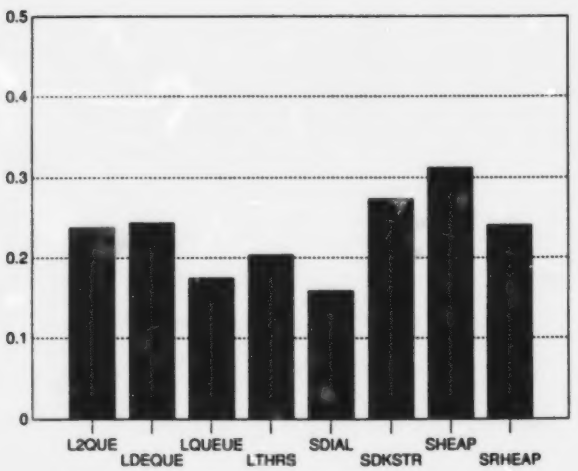


Figure 4.8 : Pourcentage d'amélioration des temps d'exécution de la figure 4.7

SHEAP est celui qui profite le plus de l'usage des pointeurs avec un gain de 31 %. Remarquons que pour les réseaux complets, la file de priorité comprend exactement n-k sommets à l'itération k. Ainsi, le travail de mise à jour du tas ("siftup, sifdown", voir Tarjan (1983)) nécessite un grand nombre d'accès aléatoires aux tableaux. Pour cette raison, l'usage des pointeurs est bénéfique. Malgré ce gain appréciable, SHEAP est toujours dominé par SRHEAP. Ces considérations s'appliquent aussi à SDKSTR, qui requiert le parcours d'une liste de n-k éléments à l'itération k.

Parmi les algorithmes ayant recours au tri par compartiments, SRHEAP est plus accéléré par l'usage des pointeurs que SDIAL. Signalons que dans ce dernier l'opération la plus longue est le balayage des compartiments. Or, en C, le parcours séquentiel d'un tableau avec un curseur est très efficace et le gain obtenu par l'usage des pointeurs est marginal. Pour SRHEAP, l'opération qui prend le plus de temps est la redistribution des sommets. Cela se produit à deux endroits dans l'algorithme : amélioration des étiquettes et recherche du sommet minimum. Puisque cette opération requiert plusieurs accès aléatoires aux tableaux, l'usage des pointeurs a pour effet d'améliorer les performances de cet algorithme.

4.8.2 Réseau aléatoire

Nous avons généré quatre réseaux aléatoires différents, dont les tailles sont les suivantes (n=1000, m=10 000), (n=1000, m=30 000), (n=3000, m=10 000) et (n=3000, m=30 000). La valeur du paramètre **x2** utilisée est 0,25, telle que proposée par Glover et al. (1985a, 1985b).

Les tableaux 4.4 et 4.5 contiennent les temps d'exécution pour les deux types d'implantation. Dans tous les cas, l'algorithme LTHRS est le plus rapide. En second lieu se trouve SDIAL, sauf pour le réseau avec 3000 noeuds et 10 000 arcs où les algorithmes usant des listes sont plus rapides. Notons que c'est le réseau le moins dense. Nous remarquons que les

algorithmes LQUEUE, LDEQUE et L2QUE sont très affectés par le nombre d'arcs. Cela s'explique par le plus grand nombre de chemins alternatifs qui existent dans les réseaux plus denses. Nous remarquons aussi que SDKSTR est une procédure très lente pour les réseaux aléatoires. Mentionnons que ces conclusions sont aussi celles obtenues par Gallo et Pallottino (1988) pour ce type de réseau.

Tableau 4.4 : Réseau aléatoire, implantations traditionnelles

ALGORITHMES	n=1000 m=10 000	n=1000 m=30 000	n=3000 m=10 000	n=3000 m=30 000
	lmax = 100			
L2QUE	192	625	222	679
LDEQUE	189	618	223	642
LQUEUE	180	520	207	592
LTHRS	**115**	**242**	**197**	**333**
SDIAL	138	280	217	413
SDKSTR	2341	2662	12011	20050
SHEAP	240	397	568	790
SRHEAP	183	316	425	581
	lmax = 10 000			
L2QUE	197	740	224	699
LDEQUE	195	714	234	683
LQUEUE	190	598	244	679
LTHRS	**114**	**258**	**211**	**374**
SDIAL	182	293	351	452
SDKSTR	2336	2832	12045	20179
SHEAP	237	393	581	744
SRHEAP	304	426	831	914

SHEAP est toujours plus rapide que SRHEAP quand **lmax** est 10 000 tandis que l'inverse se produit pour **lmax** valant 100. Lorsque la valeur de **lmax** est 100, le nombre de compartiments nécessaires à SRHEAP est 9. De plus, la probabilité élevée que plusieurs arcs aient la même

Tableau 4.5 : Réseau aléatoire, implantations avec des pointeurs

ALGORITHMES	n=1000 m=10 000	n=1000 m=30 000	n=3000 m=10 000	n=3000 m=30 000
	lmax = 100			
L2QUE	138	496	167	537
LDEQUE	130	473	146	482
LQUEUE	135	397	163	451
LTHRS	**88**	**178**	**144**	**244**
SDIAL	103	222	171	314
SDKSTR	1639	1889	8423	14154
SHEAP	202	333	488	655
SRHEAP	156	247	355	438
	lmax = 10 000			
L2QUE	152	567	187	556
LDEQUE	147	572	168	555
LQUEUE	142	429	159	466
LTHRS	**84**	**191**	**141**	**258**
SDIAL	147	233	335	371
SDKSTR	1640	2009	8046	14303
SHEAP	197	334	488	660
SRHEAP	248	345	662	732

longueur a pour effet qu'un très grand nombre de sommets se retrouvent dans le premier compartiment. Par conséquent, le nombre de transferts est moins élevé. Notons que SHEAP n'est pas affecté par la longueur du plus long arc.

La procédure par ajustements sélectifs la plus efficiente est SDIAL. Aussi, dans les deux types d'implantation en C, cet algorithme est peu affecté par la valeur de **lmax**, sauf pour le réseau le moins dense (n=3000, m=10 000).

Dans tous les cas, le classement des performances des algorithmes n'est pas modifié par l'usage des pointeurs. Ceci est illustré à la figure 4.9 pour le cas du réseau aléatoire de 1000 sommets et 10 000 arcs. Les algorithmes par ajustements sélectifs sont environ 23 % plus rapides quand

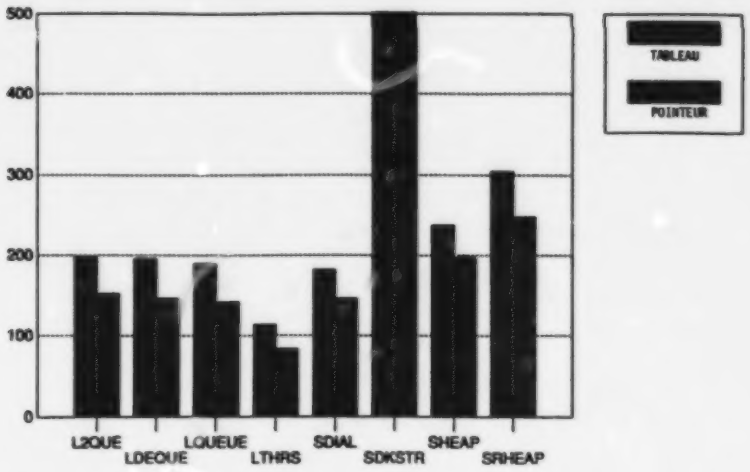


Figure 4.9 : Réseau aléatoire : lmax=10 000, n=1000, m=10 000

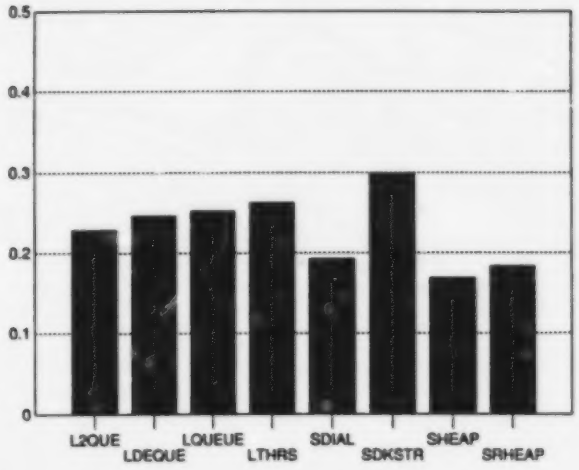


Figure 4.10 : Pourcentage d'amélioration des temps d'exécution à la figure 4.9

leur implantation fait usage des pointeurs. L'algorithme le plus amélioré est SDKSTR. Cependant, les mauvaises performances de ce dernier font en sorte que cela ne contribue pas à le rendre compétitif avec les autres programmes. Finalement, les algorithmes SDIAL, SHEAP et SRHEAP montrent une économie de 18 % du temps de calcul.

4.8.3 Réseau grille

Pour ce dernier type, nous avons généré quatre réseaux de 2500 noeuds et environ 10 000 arcs en faisant varier le nombre de rangées (p) et le nombre de colonnes (q) de la grille. Pour ces réseaux, Glover et al. (1985a, 1985b) emploient 1,5 pour la valeur de **x2** dans LTHRS. Mentionnons que Gallo et Pallottino (1988) ont testé plusieurs valeurs pour ce paramètre et ils suggèrent de fixer x2 à 1. Toutefois, nos expérimentations ont été menées avec 1,5.

Les résultats numériques sont présentés aux tableaux 4.6 et 4.7. Comme précédemment, l'usage des pointeurs n'a pas modifié le classement, ni le comportement des algorithmes. Ainsi, pour les deux types d'implantation, les algorithmes L2QUE, LDEQUE et LTHRS sont les plus efficients pour ce type de réseau. Mentionnons que ces résultats sont conformes à ceux trouvés par Gallo et Pallottino (1988).

Le temps pris par LQUEUE augmente énormément quand le nombre de colonnes augmente pendant que le nombre de rangées diminue. Cela s'explique par le fait que LQUEUE parcourt le graphe niveau par niveau. Ainsi, quand q >> p, le nombre de niveaux à analyser est très grand.

Nous observons aussi que le temps pris par SDIAL augmente quand q est plus grand que p. Rappelons que si **dmax** vaut la longueur du plus long chemin le plus court, c'est-à-dire

$$\mathbf{dmax} = \max_{i \in N} \{\Pi_i^*\},$$

Tableau 4.6 : Réseau grille, implantations traditionnelles

ALGORITHMES	p=50 q=50	p=25 q=100	p=10 q=250	p=5 q=500
	lmax = 100			
L2QUE	**176**	**165**	**169**	**157**
LDEQUE	**158**	**165**	**164**	**159**
LQUEUE	318	592	1236	2216
LTHRS	**178**	**166**	**166**	**164**
SDIAL	196	210	229	247
SDKSTR	1228	812	383	297
SHEAP	391	372	318	270
SRHEAP	440	447	484	495
	lmax = 10 000			
L2QUE	**172**	**184**	**171**	**162**
LDEQUE	**169**	**169**	**167**	**159**
LQUEUE	357	615	1425	2211
LTHRS	**169**	**169**	**166**	**155**
SDIAL	829	1080	2480	4195
SDKSTR	1213	827	405	298
SHEAP	378	377	318	288
SRHEAP	727	727	732	726

alors SDIAL doit balayer **dmax** compartiments. Dans la situation où **lmax** vaut 10 000 et q >> p, **dmax** est très grand, affectant ainsi le temps pris par SDIAL pour résoudre le problème. Toutefois, nous constatons que dans tous les cas, la valeur de **lmax** affecte le rendement de SDIAL. Cependant, nous remarquons que l'effet de **lmax** sur SDIAL est moins prononcé pour les programmes en C que pour les programmes FORTRAN de Gallo et Pallottino (1988). Afin de comparer leurs résultats avec les nôtres, nous avons calculé le ratio suivant :

$$\frac{(\text{temps pris avec } \mathbf{lmax} = 10\,000) - (\text{temps pris avec } \mathbf{lmax} = 100)}{(\text{temps pris avec } \mathbf{lmax} = 100)}$$

Tableau 4.7 : Réseau grille, implantations avec des pointeurs

ALGORITHMES	p=50 q=50	p=25 q=100	p=10 q=250	p=5 q=500
	lmax = 100			
L2QUE	**128**	**125**	**128**	**120**
LDEQUE	**114**	**114**	**116**	**108**
LQUEUE	241	450	958	1535
LTHRS	**124**	**124**	**121**	**121**
SDIAL	160	157	168	181
SDKSTR	861	587	278	205
SHEAP	325	314	255	222
SRHEAP	327	364	377	399
	lmax = 10 000			
L2QUE	**129**	**137**	**121**	**122**
LDEQUE	**129**	**120**	**114**	**116**
LQUEUE	256	417	981	1584
LTHRS	**120**	**124**	**119**	**116**
SDIAL	706	923	2211	3708
SDKSTR	886	585	289	208
SHEAP	328	308	282	253
SRHEAP	583	582	595	591

Cela donne une mesure pour évaluer l'effet de **lmax** qui est indépendante des ordinateurs. Toutefois, mentionnons que nous supposons que leurs réseaux sont les mêmes que les nôtres. Les valeurs sont dans le tableau 4.8. Clairement, le parcours séquentiel d'un tableau est plus efficace en C qu'en FORTRAN. De plus, il semble que cela soit plus rapide avec les faux-pointeurs qu'avec les véritables pointeurs. À notre avis, cela peut dépendre du test qui est effectué pour le parcours circulaire des compartiments.

Tableau 4.8 : Effet de **lmax**

	p=50 q=50	p=25 q=100	p=10 q=250	p=5 q=500
FORTRAN	5,56	7,54	17,49	30,25
POINTEUR	3,41	4,88	12,16	19,49
TABLEAU	3,23	4,14	9,83	15,98

L'algorithme SHEAP est toujours meilleur que SRHEAP, l'écart entre les deux grandissant avec **lmax**. Nous remarquons aussi que le temps pris par SHEAP et SDKSTR diminue lorsque q devient plus grand que p, car le nombre de sommets dans la file de priorité, à chaque itération, est moins élevé.

La figure 4.11 compare les temps d'exécution des algorithmes par un réseau grille de 50 colonnes et 50 rangées quand la valeur de **lmax** est 10 000. Nous constatons la nette dominance des algorithmes L2QUE, LDEQUE et LTHRS. L'algorithme par ajustements sélectifs le plus rapide est SHEAP. La figure 4.12 montre que les implantations usant des listes sont les plus

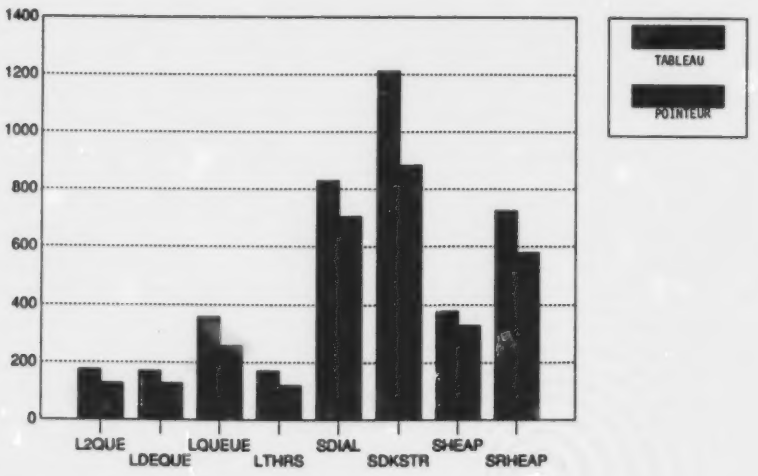


Figure 4.11 : Réseau grille, lmax=10 000, p=50, q=50

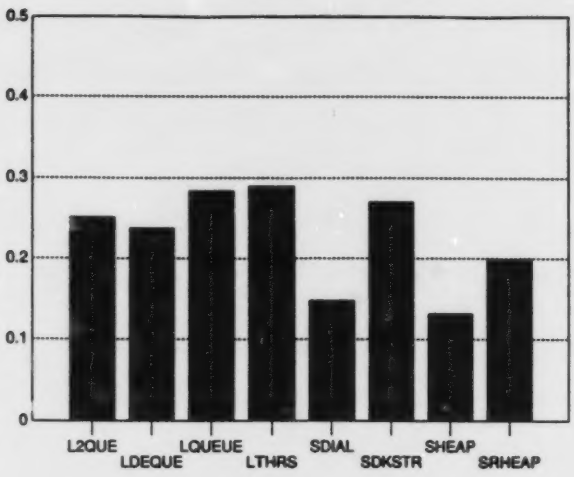


Figure 4.12 : Pourcentage d'amélioration des temps d'exécution de la figure 4.11

avantagées par l'usage des pointeurs. Soulignons que la densité d'un réseau grille est au plus 4. Les réseaux peu denses sont caractérisés par le nombre peu élevé de sommets dans la file de priorité à chaque itération. Ainsi SHEAP est peu amélioré par l'usage des pointeurs car peu d'accès aux tableaux est nécessaire.

4.8.4 Conclusions

Du point de vue de l'efficacité des algorithmes, LTHRS est la procédure la plus efficiente pour tous les réseaux que nous avons testés. Cela dépend du fait que les paramètres utilisés pour le calcul de TINC sont bien calibrés. Nous référons le lecteur à Divoky (1987) pour l'étude et la mise au point d'autres fonctions avec Seuil. Dans nos tests, l'usage des pointeurs accélère LTHRS d'au moins 20 % du temps d'exécution.

Les algorithmes LDEQUE et L2QUE se comportent de la même façon. Ils sont très rapides pour les réseaux très peu denses, les réseaux grilles ($\frac{m}{n} < 4$) par exemple. Cependant L2QUE est préférable à LDEQUE car sa complexité théorique est bornée par un polynôme, pas LDEQUE. Ces deux algorithmes produisent de meilleurs résultats que LQUEUE pour les réseaux peu denses

tandis que l'inverse se produit pour les réseaux complets. Rappelons que LQUEUE visite les noeuds et les arcs niveau par niveau et que le nombre de niveaux diminue quand la densité augmente. Ces trois algorithmes sont très faciles à mettre en oeuvre et la version qui utilise les pointeurs permet une économie du temps d'exécution d'au moins 20 % pour tous les réseaux.

SDKSTR est un algorithme usant d'une file de priorité très simple à implanter. Toutefois, la plupart du temps cette méthode est très lente, malgré l'usage des pointeurs, les pires cas étant les réseaux aléatoires.

L'algorithme SHEAP est celui qui prend le plus avantage de l'utilisation des pointeurs dans le cas des réseaux complets de plus de 100 sommets, soit 31 % d'amélioration. Cela le rend légèrement meilleur que LTHRS pour ce cas. Cela s'explique par le très grand nombre d'accès aux tableaux nécessaires pour effectuer la mise à jour du tas. Parmi les procédures par ajustements sélectifs, SHEAP domine les autres pour les réseaux grilles seulement. Notons que les performances des algorithmes usant des files de priorité sont semblables quand ils sont appliqués sur des réseaux complets.

Nous remarquons que l'efficacité de SDIAL augmente avec la densité. Cela provient du fait que SDIAL doit balayer **dmax** compartiments où

$$dmax = Max \{\Pi_i^*\}$$

Quand le réseau est très dense, la longueur du plus long chemin le plus court, **dmax**, est petite car le chemin est formé de peu d'arcs. Ainsi, pour ces réseaux, **lmax** a peu d'effet. De plus, quand l'effet est prononcé (réseau peu dense), celui-ci est moindre qu'en FORTRAN. Nous constatons aussi que SDIAL est peu amélioré par l'usage des pointeurs, par rapport aux autres algorithmes, environ 15 %, car le langage C permet le balayage efficient d'un tableau avec des faux-pointeurs.

SRHEAP est aussi un algorithme dont la file de priorité emploie des compartiments. Contrairement à SDIAL, le nombre de compartiments est très petit, 16 quand **lmax** vaut 10 000, par exemple. Bien que SRHEAP possède une très bonne borne théorique, en pratique, pour les réseaux peu denses, il est dominé par les autres méthodes qui procèdent par ajustements sélectifs. Toutefois, pour les réseaux complets, SRHEAP est aussi rapide que les autres. À notre avis, l'inefficacité de SRHEAP réside dans le nombre élevé de transferts des sommets entre les compartiments. Cela se produit à deux endroits. Premièrement, lorsqu'un sommet est amélioré, celui-ci est possiblement transféré à un nouveau compartiment. Notons que le nombre de réinsertions à cette étape de l'algorithme n'est pas plus grand que dans SDIAL. En second lieu, à l'étape de recherche du sommet minimum, lorsque les deux premiers compartiments sont vides, nous redistribuons les sommets du premier compartiment non vide dans les précédents. Ainsi, lorsque la valeur de **lmax** est grande, la probabilité que plusieurs sommets aient la même priorité est petite. Dans ce cas, quand nous redistribuons ces sommets, peu se retrouvent dans les deux premiers compartiments, augmentant ainsi le travail à effectuer. De même, pour les réseaux peu denses, le nombre de sommets dans la file de priorité, à chaque itération, est faible, réduisant la probabilité que plusieurs sommets aient la même étiquette.

Enfin, soulignons que l'usage des pointeurs permet une économie d'au moins 15 % du temps d'exécution des algorithmes du chemin le plus court par rapport aux implantations usant des tableaux et des faux-pointeurs. Cela est certainement plus grand si nous comparons ces implantations avec des programmes écrits en FORTRAN ou en PASCAL. De plus, nous avons observé des gains allant jusqu'à 30 % dans certains cas. Aussi, nous remarquons que pour la plupart des algorithmes, les implantations usant des pointeurs ne sont pas plus compliqués que les mises en oeuvre traditionnelles.

CHAPITRE 5

CHEMINS LES PLUS COURTS ENTRE TOUS LES COUPLES DE SOMMETS

À première vue, le problème du chemin le plus court entre tous les couples de sommets semble simple. En effet, pour le solutionner il suffit d'appliquer un algorithme présenté au chapitre 3 à chacun des sommets du réseau. Lorsque les longueurs des arcs sont non négatives, cela requiert au plus $O(nm\log(n))$ calculs avec l'utilisation d'une représentation en file de priorité. Dans le cas où cette condition n'est pas remplie, la complexité théorique est $O(mn^2)$. L'intérêt de cette méthode repose sur sa facilité de mise en oeuvre. D'ailleurs c'est pour cela que peu d'auteurs ont cherché à la résoudre autrement.

Dans ce chapitre, nous explorons des alternatives à cette méthode facile d'implantation et nous les comparons à l'aide d'expérimentations numériques. La motivation de cette recherche provient du fait qu'après avoir calculé une arborescence des chemins les plus courts issus d'un sommet r, l'arborescence suivante, dont la racine est s, partage souvent plusieurs arcs avec la précédente. Cela nous suggère d'employer des méthodes de réoptimisation, basées sur l'algorithme dual du simplexe, pour accélérer le processus. De plus, à notre connaissance, il n'existe aucun article rapportant plusieurs résultats d'expérimentations numériques pour ces méthodes.

Nous commençons ce chapitre en énonçant quelques propositions concernant la transformation des longueurs des arcs. Ceci nous permet d'écrire une procédure séquentielle efficace pour la résolution du problème du chemin le plus court entre tous les couples de sommets d'un réseau quand les longueurs des arcs sont quelconques. Par la suite, nous présentons les résultats fondamentaux sur lesquels repose la spécialisation de la méthode duale du simplexe pour la réoptimisation des arborescences des chemins les plus courts. Nous donnons la description de deux algorithmes qui s'appuient sur cette manière de faire. Nous terminons en discutant de quelques résultats numériques.

Dans notre discussion, nous omettons volontairement les algorithmes par calcul matriciel. Ceux-ci sont très élégants et ils possèdent une borne théorique $O(n^3)$. Il est ainsi possible, par exemple, de mettre en oeuvre l'algorithme de Floyd (1962) en six lignes de FORTRAN. Malheureusement, l'espace mémoire requis est $O(n^2)$. Ceci explique pourquoi ils ne sont pas utilisés en pratique. En revanche, nous proposons au lecteur intéressé par ce sujet de consulter les ouvrages de Floyd (1962), Warshall (1962), Dantzig (1966) et Grassin et Minoux (1973).

5.1 Transformation des longueurs des arcs

Dans cette section, nous rappelons deux propositions (voir Gallo, 1979) concernant la transformation des longueurs des arcs. Notons par T^*_r, une arborescence optimale, dont la racine est le sommet r, pour le problème du chemin le plus court. Les longueurs optimales des chemins sont Π^*_{rj}, pour tout $j \in N$. La première proposition stipule qu'il existe un calcul qui permet de substituer les longueurs initiales des arcs ℓ_{ij} par de nouvelles longueurs $\bar{\ell}_{ij}$ pour lesquelles l'arborescence T^*_s, $s \neq r$, demeure optimale. La seconde proposition établit que les nombres $\bar{\ell}_{ij}$ sont non négatifs.

Proposition 5.1

Si T^*_s est une arborescence des chemins les plus courts lorsque le coût des arcs est ℓ_{ij}, $(i,j) \in U$, alors T^*_s demeure optimale lorsque le coût des arcs est remplacé par $\bar{\ell}_{ij} = \ell_{ij} + \Pi^*_{ri} - \Pi^*_{rj}$.

Preuve Soit $h_{st} = (s = t_0, t_1, \ldots, t_k = t)$, un chemin allant de s à t. La longueur totale de ce chemin avec les coûts $\bar{\ell}_{ij}$ tels que posés ci-haut est

$$\bar{d}(h_{st}) = \sum_{(i,j) \in h_{st}} \bar{\ell}_{ij} = \sum_{(i,j) \in h_{st}} (\ell_{ij} + \Pi^*_{ri} - \Pi^*_{rj})$$

$$= \sum_{(i,j) \in h_{st}} + \Pi^*_{rs} - \Pi^*_{rt_1} + \Pi^*_{rt_1} - \Pi^*_{rt_2} \ldots - \Pi^*_{rt}$$

$$= d(h_{st}) + \Pi^*_{rs} - \Pi^*_{rt} .$$

Donc, pour tout chemin de s à t, l'ancienne et la nouvelle longueur diffèrent de $\Pi^*_{rs} - \Pi^*_{rt}$. Puisque cette constante est indépendante du chemin h_{st}, le chemin le plus court au sens des nouvelles longueurs est aussi un chemin le plus court avec les longueurs originales.

∎

Un corollaire immédiat est que

$$\bar{\Pi}^*_{sj} = \Pi^*_{sj} + \Pi^*_{rs} - \Pi^*_{rj}$$

est la plus courte distance entre les sommets s et j avec les longueurs transformées.

Proposition 5.2

$$\bar{\ell}_{ij} = \ell_{ij} + \Pi^*_{ri} - \Pi^*_{rj} \geq 0 \qquad \text{pour tout } (i,j) \in U$$

Preuve Supposons qu'il existe un arc (i,j) pour lequel $\bar{\ell}_{ij}$ est plus petit que zéro. En substituant $\bar{\ell}_{ij}$ par sa valeur, nous avons

$$\ell_{ij} + \Pi^*_{ri} - \Pi^*_{rj} < 0 ,$$

ou bien d'une manière équivalente

$$\Pi^*_{rj} > \ell_{ij} + \Pi^*_{ri} .$$

Ceci est une contradiction car Π^*_{rj} est la distance la plus courte entre les sommets r et j.

∎

Remarquons que cette transformation revient à utiliser le coût réduit des arcs à la place des longueurs originales (voir prochaines sections).

Il est intéressant de souligner que ces deux propositions nous indiquent un moyen de construire une procédure séquentielle nécessitant au plus O(nmlog(n)) opérations pour le calcul des chemins les plus courts entre tous les couples de sommet même lorsque les longueurs des arcs sont quelconques. En effet, après avoir calculé la première arborescence T_r^* avec un algorithme qui permet de traiter le cas des longueurs négatives (e.g. LQUEUE, LDEQUE ou L2QUE), nous appliquons un algorithme par ajustements sélectifs pour déterminer les autres arborescences après avoir rendu les coûts non négatifs avec la formule

$$\ell_{ij} = \ell_{ij} + \Pi_{ri}^* - \Pi_{rj}^* \ .$$

La procédure est la suivante :

Procédure 5.1 Chemins les plus courts entre tous les couples de sommet

1. Initialisation

$S \leftarrow N$,

choisir un sommet r dans S,

$S \leftarrow S - \{r\}$,

déterminer T_r^* et Π_{rj}^*, pour tout $j \in N$, avec un algorithme qui permet de traiter le cas des longueurs négatives.

2. Test d'arrêt

Si $S = \emptyset$ alors Terminer, Sinon,

choisir un sommet s dans S,

$S \leftarrow S - \{s\}$.

3. Transformation des longueurs

Poser $\ell_{ij} \leftarrow \ell_{ij} + \Pi_{ri}^* - \Pi_{rj}^*$ pour tout $(i,j) \in U$.

4. Calcul de l'arborescence des chemins les plus courts issus de s

Déterminer T_s^* et $\bar{\Pi}_{sj}^*$, pout tout $j \in N$, avec un algorithme de recherche du minimum d'abord.

5. Longueurs initiales

Poser $\Pi_{si}^* = \bar{\Pi}_{si}^* - \Pi_{rs}^* + \Pi_{ri}^*$, pour tout $i \in N$,

$r \leftarrow s$,

aller à l'étape 2.

Dans cette procédure, si l'on désire seulement évaluer les chemins les plus courts issus des sommets faisant partis d'un sous-ensemble de N, nous remplaçons l'opération "S ← N" par "S ← O" où O est un sous-ensemble de N.

Dans les prochaines sections, nous étudions des algorithmes de réoptimisation, basés sur la méthode duale du simplexe, pour calculer une arborescence T_s^* à partir d'une arborescence T_r^*.

5.2 Concepts fondamentaux

Au chapitre 3, nous avons vu que le problème des chemins les plus courts émanant d'un sommet r s'interprète comme un problème de détermination du flux à coût minimum dans un réseau. En plus de pouvoir solutionner le programme linéaire qui en résulte par la méthode primale du simplexe, nous avons exposé les spécialisations possibles. Ici, nous poursuivons en montrant que la recherche d'une nouvelle arborescence T_s^*, si T_r^* est connu, peut être exécutée par la méthode duale du simplexe.

Nous rappelons que le programme linéaire servant à déterminer T_r^* est :

Problème P1 $\quad \text{Min} \sum_{(i,j)\in U} \ell_{ij} x_{ij}$

s.à.

$$\sum_{(i,j)\in\Lambda_i} x_{ij} - \sum_{(j,i)\in\Lambda_i^{-1}} x_{ji} = b_i \qquad i=1,2,\ldots,n$$

$$x_{ij} > 0 \quad \text{pour tout } (i,j)\in U$$

où $\quad b_i = \begin{cases} n-1 & \text{si } i=r \\ -1 & \text{sinon} \end{cases}$

Le programme linéaire dual, P2, correspondant à P1 s'obtient en associant la variable duale $-\Pi_{ri}$ à la $i^{\text{ième}}$ contrainte.

Problème P2 $\quad \text{Max} \sum_{j=1}^{n} (\Pi_{rj} - \Pi_{rr})$

s.à.

$$\ell_{ij} + \Pi_{ri} - \Pi_{rj} > 0 \qquad \text{pour tout } (i,j)\in U$$

Puisqu'il y a une équation redondante dans les contraintes de 1, nous fixons $\Pi_{rr} = 0$. Dans ce cas, les variables duales optimales représentent les plus courtes distances.

À partir d'une solution optimale pour P1, le calcul des chemins les plus courts issus du sommet origine s s'effectue en perturbant le membre de droite de la façon suivante :

$$b'_i = b_i \qquad \text{si } i\neq r,s$$

$$b'_r = -1$$

$$b'_s = n-1$$

Cela génère le programme linéaire P3. Dans ce cas, le programme dual associé à ce nouveau problème est

Problème P4 $\quad \text{Max} \sum_{j=1}^{n} (\Pi_{rj} - \Pi_{rs})$

s.à.

$$\ell_{ij} + \Pi_{ri} - \Pi_{rj} \geq 0 \qquad \text{pour tout } (i,j) \in U$$

Une base optimale pour P1 est l'arborescence T_r^*. Celle-ci est aussi une base pour le programme perturbé. Par contre, elle est non réalisable pour ce dernier. D'autre part, le vecteur $(\Pi_{r1}^*, \ldots, \Pi_{rn}^*)$ satisfait les contraintes du programme dual P4. Dans ces conditions, il est naturel de le résoudre par la méthode duale du simplexe.

L'algorithme dual du simplexe consiste à appliquer l'algorithme primal du simplexe au programme dual en effectuant les opérations de changement de base (pivots) sur le tableau primal. La procédure débute avec une solution de base non faisable pour le primal. Par contre, les variables duales associées à cette base doivent réaliser les contraintes du programme dual.

Mentionnons que la non-réalisabilité provient du fait que certaines variables de base ont une valeur négative. Cela se produit fréquemment après une perturbation du membre de droite. Au lieu de résoudre à nouveau avec l'algorithme primal du simplexe, des pivots sont effectués dans le but de rendre la solution initiale réalisable pour le primal, en assurant le maintien de la faisabilité des variables duales.

5.3 Résultats fondamentaux

Soit T_r^* une arborescence optimale pour le problème des chemins les plus courts dont r est le sommet racine. Le nouveau sommet origine est s et nous voulons solutionner le problème P3. Après avoir modifié le membre de droite de P1 en faisant $b_i' = b_i + s_i$ pour tout i=1,2,...,n où

$$s_i = \begin{cases} -n & i=r \\ n & i=s \\ 0 & \text{sinon,} \end{cases}$$

la nouvelle valeur des variables de base est

$$x' = B^{-1}b' = B^{-1}(b+s)$$
$$= x + B^{-1}s$$

où B est une matrice plein rang, obtenue en ajoutant un arc artificiel, représentant une base optimale pour P1. L'inverse de B est B^{-1}.

Nous rappelons que pour un problème de chemin le plus court, il y a une équivalence biunivoque (Dantzig, 1963) entre une base réalisable et une arborescence. De plus, il est inutile de connaître B^{-1} puisque le travail s'effectue directement sur le réseau.

En vue de caractériser le flux sur les arcs de base après la perturbation du membre de droite, nous procédons à l'analyse suivante.

Nous appelons T'_s, l'arbre obtenu à partir de l'arborescence optimale T^*_r lorsque le nouveau sommet racine est s. Nous insistons sur le fait que T'_s est un arbre, mais non pas une arborescence. Cette situation est illustrée à la figure 5.1.

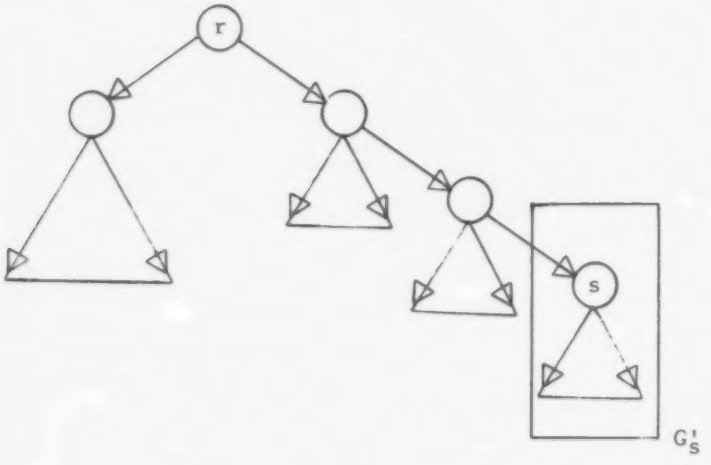


Figure 5.1 : $T'_s = T^*_r$

Considérons une chaîne (unique) allant de s à i dans T'_s. Les arcs de cette chaîne qui sont orientés vers s sont des arcs à contresens. Clairement, les arcs de la chaîne joignant les sommets s et r sont les seuls arcs à contresens dans T'_s. La présence des arcs à contresens explique pourquoi T'_s n'est pas une arborescence.

Proposition 5.3

Soit N_s, le sous-ensemble des sommets de N qui sont accessibles par un chemin à partir de s dans T'_s. Le premier résultat que nous montrons établit que les arcs du sous-graphe partiel $G'_s = (N_s, \{u \text{ tel que } u \in T'_s\})$ (voir figure 5.1) sont optimaux pour le problème P3.

Preuve Clairement, G'_s est une arborescence. D'autre part, par le principe d'optimalité, si un chemin le plus court allant de r à i traverse le sommet s, alors le sous-chemin joignant les sommets s à i est aussi un chemin minimal. Autrement dit, il existe une arborescence des chemins les plus courts issus de s, T^*_s, tel que $G'_s \subset T^*_s$.

∎

Soit M, un sous-ensemble de sommets de N et $\bar{M} = M-N$ son complément. Posons $b'_M = \sum_{i \in M} b'_i$. Nous remarquons que b'_M est plus grand que zéro si et seulement si le sommet s appartient à M. Cette partition en deux ensembles disjoints des sommets nous permet de définir la **coupe**

$$(M,\bar{M}) = \{(i,j) \in U \text{ tels que } i \in M \text{ et } j \in \bar{M}\}.$$

Le flux de cette coupe lorsque s est le sommet origine est

$$X'(M,\bar{M}) = \sum_{(i,j) \in (M,\bar{M})} x_{ij}$$

En sommant les contraintes pour tous les i dans M, nous obtenons

$$b'_M = X'(M,\bar{M}) - X'(\bar{M},M)$$

Nous symbolisons par $R_{s,u}$, l'ensemble des sommets i tel que la chaîne qui relie le sommet s à i sur T'_s ne contient pas l'arc u. En d'autres mots, $R_{s,u}$ inclut tous les sommets connexes à s lorsque l'arc u est ôté de T'_s (voir figure 5.2).

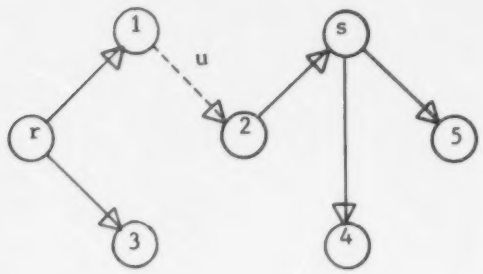


Figure 5.2 : $R_{s,u} = \{s,2,4,5\}$

Proposition 5.4

La valeur des variables de base après la perturbation du membre de droite est

$$x'_u = \begin{cases} -b'_{R_{s,u}} & \text{si u est un arc à contresens} \\ b'_{R_{s,u}} & \text{sinon} \end{cases}$$

Preuve

Posons $M = R_{s,u}$

Puisque $s \in M$ et T'_s est un arbre, nous avons

$$(M,\bar{M})\cap T'_s = \begin{cases} u & \text{si } u \text{ est orienté de } s \text{ à } \bar{M} \\ \emptyset & \text{sinon} \end{cases}$$

$$(\bar{M},M)\cap T'_s = \begin{cases} u & \text{si } u \text{ est un arc à contresens} \\ \emptyset & \text{sinon} \end{cases}$$

D'autre part, étant donné que le flux sur les arcs hors-base est nul, nous avons $b'_M = x'(M,\bar{M}) - x(\bar{M},M) =$

$$\sum_{(i,j)\in(M,\bar{M})\cap T'_s} x'_{ij} - \sum_{(i,j)\in(\bar{M},M)\cap T'_s} x'_{ij}$$

$$= \begin{cases} -x'_u & \text{si } u \text{ est un arc à contresens} \\ x'_u & \text{sinon} \end{cases}$$

∎

Corollaire 5.5

La valeur des variables de base après la perturbation du membre de droite est

$$x'_u = \begin{cases} |R_{s,u}| - n & \text{si } u \text{ est un arc à contresens} \\ n - |R_{s,u}| & \text{sinon} \end{cases}$$

Preuve Il suffit de constater que $b'_i = -1$, $i \neq s$, alors $b'_{R_{s,u}} = n - |R_{s,u}|$

∎

Ceci nous permet de conclure que les seuls arcs dont le flux est négatif sont les arcs à contresens. De plus, nous savons facilement les identifier. En effet, les seuls arcs à contresens se trouvent sur la chaîne allant de s à r dans T'_s. Par conséquent, la non-réalisabilité de la solution de base pour le primal repose uniquement sur la présence des arcs à contresens.

Après avoir déterminer la variable de sortie, le but est de passer à une autre solution de base qui est "moins" non réalisable tout en s'assurant que les variables duales demeurent faisables pour le problème P4. Alors le choix de la variable d'entrée repose sur la proposition suivante.

Proposition 5.7

L'arc d'entrée f est l'arc qui possède le coût réduit le plus faible parmi les arcs de la coupe $(N_s, \bar{N}_s)$, où N_s est l'ensemble des sommets connectés à s par un chemin dans T'_s.

Preuve Lorsque l'arc e quitte la base, l'arbre T'_s est séparé en deux composantes connexes. D'une part, il y a le sous-graphe partiel $G'_s = (N_s, \{u \text{ tel que } u \in T'_s\})$. Clairement G'_s est une arborescence. Pour sa part, le sous-graphe partiel $\bar{G}'_s = (\bar{N}_s, \{u \text{ tel que } u \in T'_s\})$ est un arbre. Dans le but de passer à une autre solution de base, l'arc f est choisi parmi les arcs de la coupe $(N_s, \bar{N}_s)$ où parmi ceux de la coupe $(\bar{N}_s, N_s)$. Or, les arcs de la coupe $(\bar{N}_s, N_s)$ sont des arcs orientés dans le même sens que e. En plus de ne pas contribuer à la non-moins-réalisabilité de la nouvelle solution de base (extension de G'_s), l'entrée d'un arc f, $f \neq e$ de la coupe $(\bar{N}_s, N_s)$ a pour conséquence de rendre le coût réduit de l'arc e négatif. Donc, l'arc f est pris parmi les arcs de la coupe $(N_s, \bar{N}_s)$, et afin de maintenir le coût réduit des arcs de cette coupe, non négatif, il doit avoir le coût réduit le plus faible (voir section 5.4).

∎

5.4 Exemple

Ceci nous permet de mettre en oeuvre une spécialisation de la méthode duale du simplexe qui prend avantage de la structure de réseau du problème. Avant de présenter formellement la procédure, nous décrivons une

itération de la méthode duale avec un exemple. Considérons le réseau de la figure 5.3. Les nombres sur les flèches représentent la longueur des arcs. Le sommet origine est 1 et nous distinguons T_r^* avec les traits en gras. De plus, pour chacun des sommets j=1,2,...,8, Π_{1j}^* est la longueur minimale pour se rendre de 1 à j.

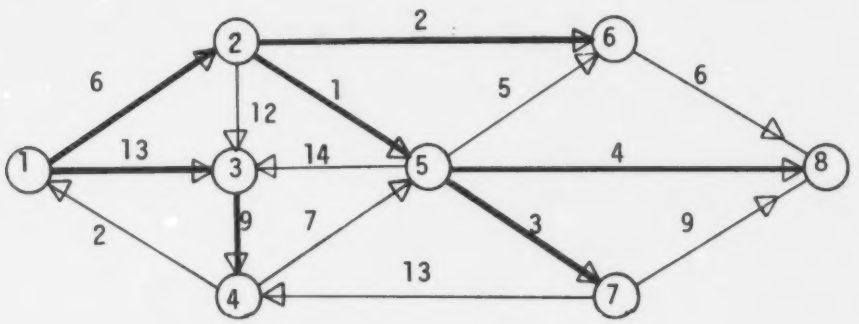


Figure 5.3 : T_r^* (r=1)

La figure 5.4 illustre la solution de base pour le problème perturbé. Le sommet 2 est le nouveau sommet origine. Les nombres sur les flèches représentent la valeur des variables de base après le changement d'origine. Ils sont uniquement déterminés en solutionnant le système :

$$\sum_{(i,j)\in\Lambda_i} T_2' \, x_{ij} - \sum_{(i,j)\in\Lambda_i^{-1}} T_2' \, x_{ji} = \begin{cases} -1 & \text{si } i=1,3,4,5,6,7,8 \\ 7 & \text{si } i=2 \end{cases} .$$

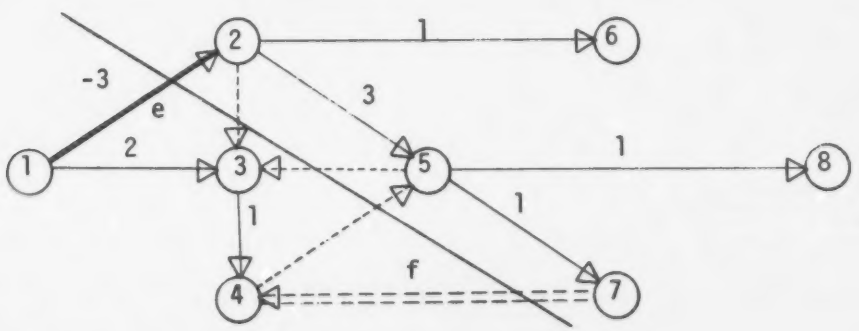


Figure 5.4 : T_s' (s=2)

L'arc de sortie est e=(1,2). C'est un arc à contresens et la valeur de son flux est plus petite que zéro. L'ensemble N_s est $\{2,5,6,7,8\}$ et $\bar{N}_s$ égale $\{1,3,4\}$. L'arc d'entrée f est celui qui possède le coût réduit le plus faible parmi les arcs de la coupe $(N_s,\bar{N}_s) = \{(2,3),(5,3),(7,4)\}$;

$$\bar{\ell}_{23} = 12 + 6 - 13 = 5$$

$$\bar{\ell}_{53} = 14 + 7 - 13 = 8$$

$$\bar{\ell}_{74} = 13 + 10 - 22 = 1 +$$

Suite au changement de base, $T_2' \leftarrow T_2' - \{(1,2)\} \cup \{(7,4)\}$, la mise à jour des variables duales consiste à déterminer une nouvelle solution au système d'équations $\Pi_{2j} = \Pi_{2i} + \ell_{ij}$ pour tous les arcs (i,j) dans T_2'. En laissant la valeur des variables duales fixes pour les sommets dans N_s, nous obtenons la valeur de Π_{24} en résolvant $\Pi_{24} \leftarrow \Pi_{27} + \ell_{74}$; $\Pi_{24} \leftarrow 10 + 13 = 23$. Dans ce cas, la nouvelle solution de base est celle de la figure 5.5.

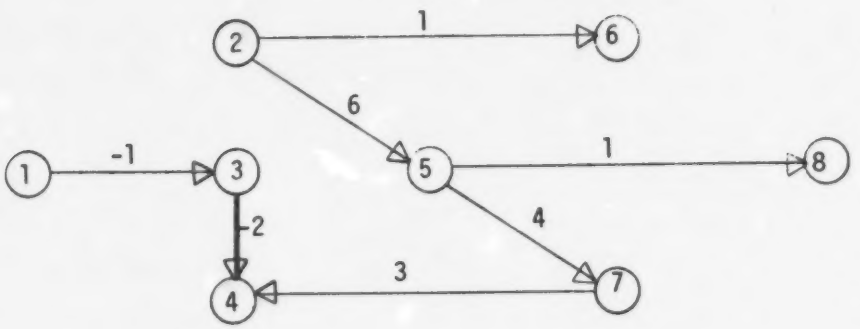


Figure 5.5 : T_s' après un pivot

D'une manière équivalente, si nous fixons la valeur des variables duales des sommets de $\bar{N}_s$, nous calculons la nouvelle valeur des variables duales des sommets de N_s en faisant

$$\Pi_{2j} \leftarrow \Pi_{2j} - \bar{\ell}_f \qquad \text{pour tout } j \text{ dans } N_s .$$

Dans l'un ou l'autre des deux cas, nous vérifions que le coût réduit des arcs est modifié seulement pour les arcs des coupes $(N_s,\bar{N}_s)$ et $(\bar{N}_s,N_s)$, et ceux-ci demeurent non négatifs. Ce n'est pas le cas si l'arc d'entrée est un arc à contresens ou si le coût réduit de l'arc d'entrée n'est pas le plus petit.

5.5 Algorithme de réoptimisation duale

Finalement, l'algorithme de réoptimisation duale est le suivant.

ALGORITHME 5.1

1. Initialisation

Soit T^*_r, une arborescence optimale pour le problème des chemins les plus courts issus du sommet r. La solution de base initiale pour le problème P3 est T'_s.

$\Pi_{sj} \leftarrow \Pi^*_{rj}$, pour tout $j \in N$. C'est une solution duale réalisable associée à T'_s.

Créer l'ensemble N_s qui contient les sommets que l'on peut atteindre par un chemin à partir de s.

$\bar{N}_s \leftarrow N - N_s$.

$z \leftarrow s$.

2. Test d'arrêt

Si N_s est vide alors TERMINER.

3. Sélection

Déterminer le premier arc à contresens, $e = (i,z)$, $i \in \bar{N}_s$.

Sélectionner l'arc f tel que

$$\ell_f = \min_{(i,j)\in(N_s,\bar{N}_s)} \{\ell_{ij} + \pi_{si} - \pi_{sj}\} .$$

4. Pivot

Mettre à jour les variables duales, $\pi_{sj} \leftarrow \pi_{sj} + \bar{\ell}_f$, pour tout $j \in \bar{N}_s$.

Mettre à jour la base, $T'_s \leftarrow T'_s - \{e\} \cup \{f\}$.

Si $f=(i,j)$ alors poser $z=j$.

Ajouter dans N_s les sommets accessibles par un chemin à partir de j,

$N_s \leftarrow N_s \cup N_j$, $\bar{N}_s \leftarrow \bar{N}_s - N_j$.

Aller à l'étape 2.

Puisqu'à chaque itération, l'ensemble N_s représente l'ensemble des sommets accessibles par un chemin à partir de s, lorsque $\bar{N}_s$ est vide, T'_s est une arborescence dont la racine est s. De plus, le choix de la variable d'entrée assure que les variables duales réalisent les contraintes du problème P4. Dans ce cas, l'algorithme termine avec une base réalisable pour le primal et une solution réalisable pour le dual. Par le théorème de dualité en programmation linéaire, nous concluons que T'_s est optimale pour le problème P3.

Proposition 5.8

L'algorithme 5.1 termine en au plus n-1 itérations.

Preuve Puisque l'arc f est choisi parmi les sommets de la coupe $(N_s,\bar{N}_s)$, à chaque itération N_s est augmenté d'au moins un sommet.

Corollaire 5.9

La complexité de l'algorithme 5.1 est $O(n^3)$.

Preuve Supposons qu'à chaque itération N_s est augmenté d'un seul sommet. Dans ce cas, la coupe $(N_s, \bar{N}_s)$ contient $k(n-k)$ arcs à la $k^{ième}$ itération. Au total, l'algorithme 5.1 nécessite $\sum_{k=1}^{n-1} k(n-k) = O(\frac{n^3}{6})$ comparaisons pour déterminer l'arc d'entrée. D'autre part, la mise à jour des variables duales requiert $\sum_{k=1}^{n-1} (n-k) = O(\frac{n^2}{2})$ opérations.

D'une manière informelle, l'algorithme précédent s'explique de la manière suivante. Après avoir calculé T_r^*, si les sommets r et s sont suffisamment proches, les deux arborescences possèdent plusieurs arcs en commun. La procédure exploite cet état de fait en commençant avec $G_s' = (N_s, \{u \text{ tel que } u \in T_s'\})$. Ensuite, nous cherchons le sommet z de $\bar{N}_s$ qui est le plus proche des sommets de N_s au sens des coûts réduits. Par la proposition 5.1, le sommet z est aussi le plus proche au sens des longueurs initiales. De plus, par le principe d'optimalité, si un chemin le plus court allant de r à k passe par le sommet z, alors le sous-chemin de z à k est aussi minimal. Donc, l'extension de G_s' consiste à greffer à G_s' la sous-arborescence du sommet z de T_r^*. Clairement, cette politique est optimale et elle s'apparente à l'algorithme de Dijkstra à la différence qu'ici G_s' est augmenté avec plus d'un arc à la fois.

Proposition 5.10

Si nous notons par $\bar{\Pi}_{sj}^*$ la longueur optimale pour se rendre du sommet s à j au sens des coûts réduits, alors

$$\bar{\Pi}_{max} = \underset{j \in N}{Max} \{\bar{\Pi}_{sj}^*\} = \bar{\Pi}_{sr}^* = \Pi_{rs}^* - \Pi_{sr}^*$$

Preuve Nous avons $\bar{\ell}_{ij} = 0$ pour tous les arcs (i,j) dans T^*_r. Par suite, $\bar{\Pi}^*_{rj} = 0$ pour tout $j=1,2,\ldots,n$. Considérons un chemin le plus court allant de s à k traversant le sommet r. La longueur de ce chemin au sens des coûts réduits est $\bar{\Pi}^*_{sr} + \bar{\Pi}^*_{rk} = \bar{\Pi}^*_{sr}$. Par conséquent, $\bar{\Pi}^*_{sr}$ est une borne supérieure pour $\bar{\Pi}^*_{sj}$. Donc, $\bar{\Pi}^*_{sj} \leq \bar{\Pi}^*_{sr}$, pour tout j dans N. En prenant j=r, ceci montre la première partie de la proposition. D'autre part, par la proposition 5.1, nous avons $\bar{\Pi}^*_{sr} = \Pi^*_{sr} + \Pi^*_{rs} - \Pi^*_{rr} = \Pi^*_{sr} - \Pi^*_{rs}$.

∎

Corollaire 5.11

L'algorithme 5.1 termine lorsque le sommet r est accroché à G'_s. Dans ce cas, G'_s correspond à T^*_s.

∎

Dans les prochaines sections, nous montrons deux spécialisations de l'algorithme 5.2 pour lesquelles la complexité théorique est $O(n^2)$. Ces algorithmes utilisent la transformation des coûts décrite au début de ce chapitre et ne requièrent pas la mise à jour des variables duales. Le premier algorithme que nous décrivons est dû à Florian et al. (1981). C'est l'algorithme DUAL. Le second, LSM ("Label setting with memory") est dû à Gallo et Pallottino (1982).

Pour la détermination des chemins les plus courts entre tous les couples de sommets avec l'un ou l'autre de ces deux algorithmes, il suffit de remplacer l'étape 4 de la procédure 5.1 par

4' Calcul de l'arborescence des chemins les plus courts issus de s

Déterminer T^*_s et $\bar{\Pi}^*_{sj}$, pout tout $j \in N$, avec l'un ou l'autre des algorithmes DUAL ou LSM.

5.6 Algorithme DUAL

Dans l'algorithme 5.1, l'étape de la détermination de l'arc d'entrée (étape 3) nécessite à chaque itération l'analyse de tous les arcs de la coupe $(N_s, \bar{N}_s)$. Cela prend beaucoup de temps, d'où la complexité $O(\frac{n^3}{6})$. Aussi, le pivot (étape 4) requiert la mise à jour de toutes les variables duales de l'ensemble $\bar{N}_s$.

Une façon efficace d'implanter l'algorithme 5.1 consiste à substituer les longueurs originales des arcs par les coûts réduits des arcs (voir Gallo, 1979; Florian et al., 1981). Nous avons discuté des propriétés de cette transformation à la section 5.1.

L'algorithme DUAL consiste à munir chacun des sommets j, $j \in N$, d'une étiquette ℓ_j et de les conserver dans une file de priorité. De plus, pour chacun des sommets j, $j \in N$, nous employons un dictionnaire, dy_j, qui contient le dernier sommet qui a contribué à diminuer l'étiquette du sommet j. Les étiquettes des sommets de l'ensemble N_s correspondent aux longueurs des chemins les plus courts issus du sommet s lorsque les longueurs des arcs sont les coûts réduits. À chaque itération, l'ensemble N_s est augmenté du sommet , $z \in \bar{N}_s$, qui possède l'étiquette dont la valeur est la plus petite ainsi que les sommets de la sous-arborescence dont z est la racine. Nous notons par T_z, la sous-arborescence dont z est la racine et par N_z les sommets de cette sous-arborescence. Les valeurs initiales des étiquettes sont :

$$\ell_j = \begin{cases} 0 & \text{pour tout } j \in N_s \\ \min_{i \in N_s} \{\ell_{ij} + \Pi^*_{ri} - \Pi^*_{rj}\} & \text{pour tout } j \in \bar{N}_s \end{cases}.$$

Ainsi, à chaque itération, la mise à jour des variables duales consiste à faire

$$\ell_j = \begin{cases} \ell_z & \text{pour tout } j \in N_z \\ \min\limits_{i \in N_z} \{\ell_j, \ell_{ij} + \Pi^*_{ri} - \Pi^*_{rj} + \ell_z\} & \text{pour tout } j \in \bar{N}_s - N_z \end{cases}.$$

Finalement, nous constatons qu'il est inutile d'identifier à chaque itération l'arc de sortie. À la place, au début de l'algorithme, nous séparons T'_s en deux parties, en enlevant l'arc (p_s,s) où P_s est le prédécesseur de S. La sous-arborescence dont la racine est le sommet s, est T_s et l'autre, dont la racine est le sommet r, est T_r. Par la suite, une itération de l'algorithme consiste à joindre à T_s la sous-arborescence T_z, $T_z \subset T_r$, où z est le sommet qui possède l'étiquette dont la valeur est la plus faible parmi les sommets de $\bar{N}_s$ (voir figure 5.6). L'algorithme termine quand le sommet r est greffé à T_s. À ce moment T_s correspond à T^*_s. Formellement, l'algorithme est le suivant :

ALGORITHME 5.2

1. Initialisation

Soit T^*_r, une arborescence optimale pour le problème des chemins les plus courts issus du sommet r et $(\Pi^*_{r1},\ldots,\Pi^*_{rn})$ le vecteur des longueurs minimales.

Enlever l'arc (p_s,s). À ce moment, nous avons les deux arborescences T_r et T_s et les deux ensembles N_s et $\bar{N}_s$.

$$\ell_j = \begin{cases} 0 & \text{pour tout } j \text{ dans } N_s \\ \min\limits_{i \in N_s} \{\ell_{ij} + \Pi^*_{ri} - \Pi^*_{rj}\} & \text{pour tout } j \text{ dans } \bar{N}_s \end{cases}.$$

$z \leftarrow s$

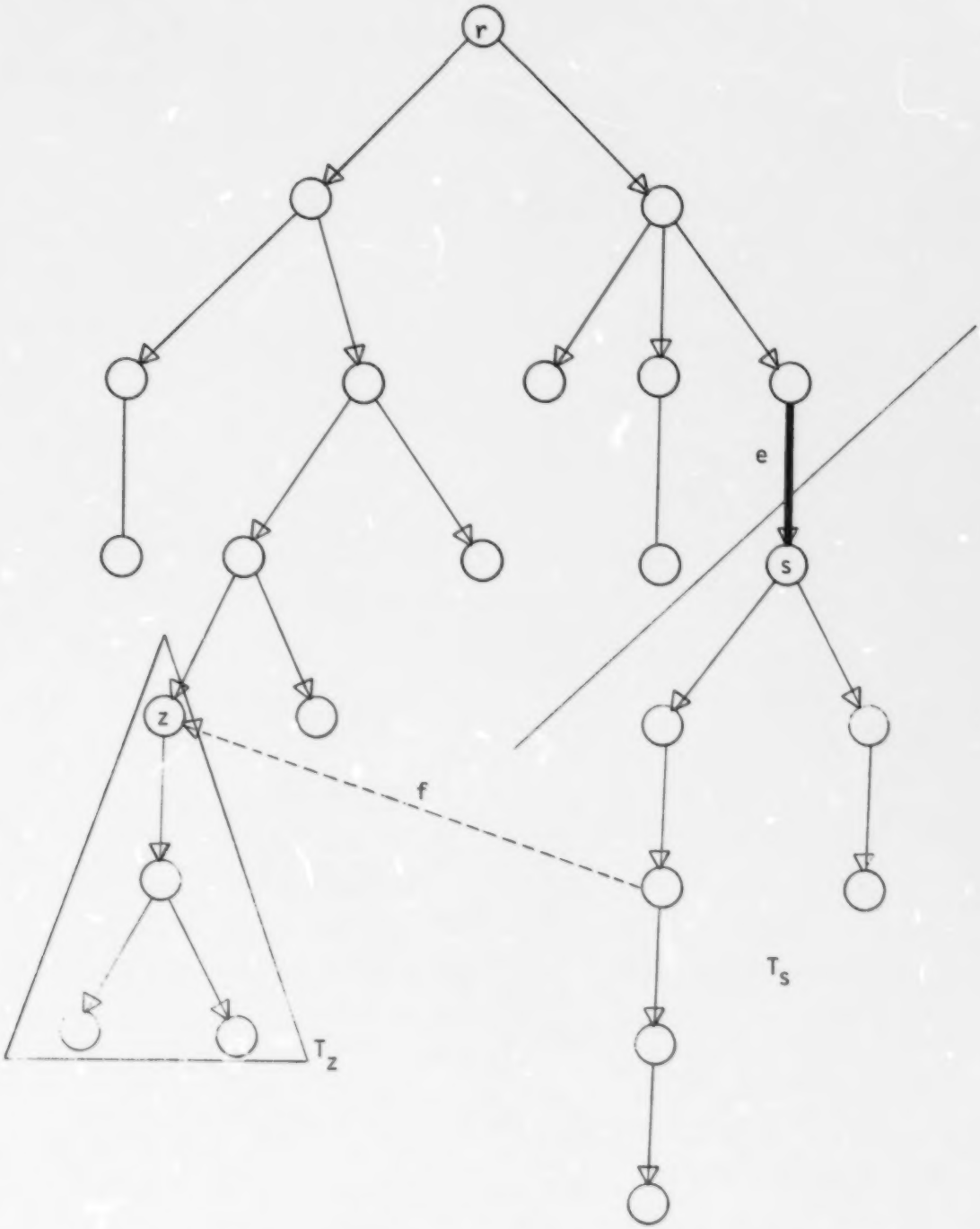


Figure 5.6 : Une itération de l'algorithme DUAL

2. Test d'arrêt

Si $\bar{N}_s$ est vide alors TERMINER

3. Sélection

$$\ell_z \leftarrow \min_{j \in \bar{N}_s} \{\ell_j\}$$

$$f \leftarrow (dy_z, z)$$

4. Extension

$$\ell_j = \begin{cases} \ell_z & \text{pour tout } j \in N_z \\ \min_{i \in N_z} \{\ell_j, \ell_{ij} + \pi^*_{ri} - \pi^*_{rj} + \ell_z\} & \text{pour tout } j \in \bar{N}_s - N_z. \end{cases}$$

$$T_s \leftarrow T_s \cup \{f\} \cup T_z$$

$$N_s \leftarrow N_s \cup N_z$$

$$\bar{N}_s \leftarrow \bar{N}_s - N_z$$

Aller à 2

Dans cette forme, l'algorithme DUAL s'interprète comme une variante de l'algorithme de Dijkstra à la différence qu'à chaque itération, possiblement, plus d'un sommet reçoivent une étiquette permanente. L'efficacité de cet algorithme est établie par la proposition suivante.

Proposition 5.12

L'algorithme DUAL requiert au plus $O(n^2)$ comparaisons.

Preuve Le pire cas correspond à effectuer n-1 itérations. À chaque pivot, la mise à jour des étiquettes des sommets de $\bar{N}_s - N_z$ nécessite l'examen des arcs de la coupe $(N_z, \bar{N}_s - N_z)$. Il y a au plus n-1 arcs

dans cette coupe à chaque pivot. Enfin, la sélection du sommet z ayant l'étiquette la plus faible requiert n-i comparaisons à l'itération i. Donc la complexité de l'algorithme DUAL est $O(n^2)$.

■

Du point de vue de la mise en oeuvre informatique, mentionnons que l'algorithme 5.1 nécessite l'utilisation de procédures pour l'exploration des arbres (e.g. visite en préordre). Florian et al. (1981) utilisent les fonctions P[], TH[] et LT[]. Pour notre part, nous employons les fonctions P[], TH[], RT[] et LT[]. Nous référons le lecteur à Pallottino et Storchi (1981,1984) et Kennington et Helgason (1980) pour une discussion approfondie sur ces fonctions. Mentionnons que la mise à jour de ces fonctions ne requièrent pas plus de $O(n^2)$ opérations élémentaires. De plus, nous avons mené nos expérimentations numériques avec deux types de file de priorité : un arbre partiellement ordonné ("heap") et une liste non triée. Les résultats numériques sont discutés à la section 5.8.

5.7 Algorithme LSM

Gallo et Pallottino (1982) ont proposé une mise en oeuvre ingénieuse de l'algorithme 5.1 qui, contrairement à l'algorithme DUAL, ne nécessite pas le parcours des arbres. L'algorithme LSM utilise le coût réduit des arcs au lieu des longueurs originales et il s'interprète comme un algorithme de marquage par extension sélective qui emploie une liste partitionnée.

Nous remarquons que le coût réduit des arcs faisant partie de l'arbre T'_s est zéro. Ce sont les arcs de base. Afin de visiter ces arcs en priorité, nous partitionnons la liste, Q, des sommets candidats (voir chapitre 3) en deux listes distinctes, Q_1 et Q_2. La liste Q_1 est une file et la liste Q_2 est une file de priorité. La règle d'insertion des sommets dans la liste des sommets candidats consiste à placer dans Q_1 les sommets

pour lesquels le coût réduit de l'arc améliorant est zéro. Les autres sommets sont insérés dans la file de priorité. Le choix à chaque itération est de prendre un sommet de Q_1. C'est seulement lorsque Q_1 est vide que nous retirons un sommet d'étiquette minimum de la file de priorité. L'algorithme termine quand les deux listes, Q_1 et Q_2 sont vides.

Formellement l'algorithme est le suivant :

ALGORITHME 5.3

1. Initialisation

Soit $(\Pi^*_{r1},\ldots,\Pi^*_{rn})$ le vecteur des longueurs des chemins les plus courts issus du sommet r.

Poser

$$\ell_j \leftarrow \begin{cases} 0 & \text{si } j=s \\ \text{infini} & \text{sinon} \end{cases}$$

$$p_j \leftarrow \begin{cases} -1 & \text{si } j=s \\ s & \text{sinon} \end{cases}$$

$Q_1 \leftarrow \{s\}$

$Q_2 \leftarrow \emptyset$

2. Sélection

si $Q_1 \neq \emptyset$ alors prendre i dans Q_1, $Q_1 \leftarrow Q_1 - \{i\}$

sinon si $Q_2 \neq \emptyset$ prendre i dans Q_2 tel que $\Pi_{si} = \underset{i \in Q_2}{\text{Min}} \{\Pi_{si}\}$

sinon Terminer

3. Extension

Pour chaque (i,j) dans A_i faire

si $\ell_i + \ell_{ij} + \Pi^*_{ri} - \Pi^*_{rj} < \ell_j$ alors

faire

$\ell_j \leftarrow \ell_i$

$p_j \leftarrow i$

si $\ell_i = \ell_j$ alors $Q_1 \leftarrow Q_1 \cup \{j\}$, $Q_2 \leftarrow Q_2 - \{j\}$

sinon $Q_2 \leftarrow Q_2 \cup \{j\}$

fin si

aller à 2

Nous soulignons que la seule différence entre les algorithmes LSM et DUAL réside dans la façon dont les sous-arborescences sont visitées. L'utilisation de la file Q_1 revient à explorer les sommets d'une sous-arborescence niveau par niveau, comme dans l'algorithme LQUEUE. Lorsque tous les sommets d'une sous-arborescence ont été visités, c'est-à-dire quand Q_1 est vide, un nouveau sommet z (voir figure 5.5) est identifié à l'aide de la file de priorité Q_2. La proposition suivante établit l'efficacité de l'algorithme LSM.

Proposition 5.13

L'algorithme LSM requiert au plus $O(n^2)$ opérations élémentaires.

Preuve À chaque itération un nouveau sommet reçoit une étiquette permanente. L'algorithme termine donc en n-1 itérations. De plus, la cardinalité de l'ensemble Q_2 est bornée par n, par conséquent, la complexité de LSM est $O(n^2)$.

5.8 Expérimentations numériques

Dans cette section, nous décrivons les expérimentations numériques que nous avons menées et nous analysons les résultats obtenus.

À notre connaissance, les deux algorithmes que nous avons décrits dans ce chapitre ont seulement été testés sur deux réseaux provenant d'une application réelle. Les détails se trouvent dans Florian et al. (1981) et Gallo et Pallottino (1982). Dans ces deux papiers, les méthodes de réoptimisation duale se sont avérées très efficientes en permettant des économies de temps de calcul allant de 20 % à 35 % par rapport à l'application séquentielle des algorithmes de D'Esopo-Pape (LDEQUE) et Dial (SDIAL).

Nous avons implanté les deux algorithmes DUAL et LSM en langage C avec des tableaux et des faux-curseurs. Nous avons employé les techniques efficaces de programmation décrites dans Gallo et Pallottino (1988) et Pallottino et Storchi (1981, 1984). Rappelons que le but que nous poursuivons dans ce chapitre consiste à comparer l'efficacité des méthodes duales par rapport à l'application séquentielle d'un algorithme décrit au chapitre 3. Pour cette raison, nous ne discutons pas des autres implantations possibles employant des pointeurs. Mentionnons aussi qu'au chapitre 4, nous avons remarqué que le classement des performances des algorithmes est peu affecté par l'utilisation des pointeurs.

Pour chacun des deux algorithmes, DUAL et LSM, nous avons testé deux files de priorité. Ainsi, DUA1 et LSM1 sont les implantations usant d'une liste simplement chaînée tandis que DUA2 et LSM2 emploient un tas. Une disquette contenant ces programmes est disponible. Le lecteur intéressé peut en faire la demande en s'adressant à l'auteur.

Comme précédemment, nous avons considéré trois types de réseaux : complets, alétoires et grilles. Les longueurs des arcs sont des nombres entiers qui proviennent d'une distribution uniforme sur l'ensemble

[1-10 000]. Également, le temps CPU est mesuré en 1/60 seconde et il représente le temps total pris pour la résolution des n-1 arborescences des chemins les plus courts. Notons que le calcul de la première arborescence ne peut pas être effectué à l'aide d'une procédure de réoptimisation. Pour cette raison, nous ne considérons pas le temps nécessaire à ce travail. Afin de comparer l'approche de réoptimisation duale à l'approche séquentielle, nous avons choisi LTHRS. Rappelons que ce dernier a été le meilleur sur la plupart des réseaux que nous avons testés au chapitre quatre.

Les tableaux 5.1, 5.2 et 5.3 montrent les résultats obtenus. Dans tous les cas, l'algorithme LTHRS est le plus rapide. Pour les réseaux complets et grilles, DUA1 est la procédure de réoptimisation duale la plus

Tableau 5.1 : Réseau complet

ALGORITHMES	n=25	n=50	n=75	n=100
DUA1	14	121	401	942
DUA2	20	137	428	982
LSM1	18	138	445	1065
LSM2	14	130	413	963
LTHRS	12	97	318	730

Tableau 5.2 : Réseau aléatoire

ALGORITHMES	n=500 m=1500	n=500 m=5000	n=500 m=10000	n=500 m=15000
DUA1	4990	15103	19002	22783
DUA2	3286	6604	9145	11394
LSM1	6981	22254	26125	30380
LSM2	2607	5328	7694	9944
LTHRS	1549	2725	4402	5760

Tableau 5.3 : Réseau grille

ALGORITHMES	p=24 q=24	p=12 q=48	p=8 q=72	p=4 q=144
DUA1	2453	2261	2167	1984
DUA2	4025	3594	3392	2934
LSM1	2986	2715	2534	2337
LSM2	3207	3320	3122	2653
LTHRS	2233	2237	2158	1931

efficiente tandis que LSM2 est, parmi ces procédures, la plus efficace pour les réseaux aléatoires.

Donc, nous remarquons que nos résultats numériques contredisent ceux trouvés dans la littérature scientifique. Cela s'explique par le fait que l'ordre dans lequel sont visités les sommets pour le calcul des arborescences influence l'efficacité des procédures de réoptimisation. Ainsi, dans le cas des applications réelles,les auteurs mentionnent qu'une analyse du réseau, a priori, a été conduite afin de déterminer la séquence de visite des sommets. Pour notre part, nous avons simplement pris les sommets dans l'ordre lexicographique, c'est-à-dire 1,2,...,n.

Une conclusion à laquelle nous parvenons est l'importance du choix de la séquence de visite des sommets. Ainsi, quand deux arborescences sont "proches", lorsqu'elles partagent plusieurs arcs en commun, par exemple, nous avons avantage à utiliser une procédure de réoptimisation. Le problème consiste à déterminer une "bonne" séquence et éventuellement, la "meilleure". Bien que ce travail soit très intéressant, il dépasse largement les cadres de ce mémoire. Pour cette raison, nous n'avons pas exploré en profondeur ces avenues de recherche. Cependant, nous apportons quelques remarques supplémentaires concernant ce problème et nous mentionnons quelques suggestions.

La notion de "bonne" ou "meilleure" séquence peut être définie a priori ou elle peut être adaptive. Une façon de déterminer a priori une bonne séquence consiste à inspecter manuellement le réseau considéré. Une seconde approche est d'appliquer un algorithme exact ou heuristique pour le problème du commis voyageur. D'autre part, une séquence adaptive consiste à choisir au fur et à mesure le prochain sommet origine. Ce dernier peut être le sommet le plus proche du sommet origine précédent, par exemple.

CHAPITRE 6

STRUCTURE DES DONNÉES

Une des principales caractéristiques des systèmes de transport des marchandises est la très grande quantité d'informations nécessaires pour en décrire le fonctionnement. C'est ainsi que, par exemple, pour une seule étude de planification tactique pour un réseau ferroviaire, Crainic et Rousseau (1986) ont considéré un réseau physique de 107 noeuds et 190 liens, 415 services potentiels, trois modes et 2 613 marchés dont les flux de marchandises empruntent de 4 000 à 5 000 itinéraires.

De plus, dans un contexte de planification, cette quantité d'informations est significativement multipliée par les diverses hypothèses concernant l'état du système de transport : réseau physique, services, demande, etc. Enfin, mentionnons que le problème de la manipulation des informations ne réside pas que dans la quantité des données, mais aussi dans leur structure de "chemin" relativement complexe.

Au cours de ce chapitre, nous discutons des structures de données nécessaires à la mise en oeuvre d'un outil interactif-graphique adapté au problème de la planification tactique et stratégique du transport des marchandises. En premier lieu, nous énumérons les principaux éléments que nous devons manipuler ainsi que leur organisation logique. Nous traitons aussi des structures de données utilisées par les différents programmes d'optimisation et d'affichage. Enfin, nous discutons des difficultés liées à l'implantation d'une banque de données pour accéder et conserver ces éléments.

Nous faisons remarquer que les objectifs de ce mémoire de maîtrise ne visent pas à produire un logiciel complet pour la planification tactique et stratégique des systèmes de transport des marchandises. Nous cherchons plutôt à établir les conditions pour qu'un tel logiciel soit opérationnel

et nous créons les fondations pour les travaux futurs. Pour ces raisons nous n'avons pas mis en oeuvre toutes les structures de données nécessaires à l'outil TACTIC. Cependant, nous soulevons les problème importants liés à l'organisation des données, et nous suggérons des avenues de recherche pour les solutionner.

6.1 Organisation logique

Au cours du chapitre premier, nous avons décrit les éléments intervenant lors du processus lors de la planification tactique et stratégique des systèmes de transport des marchandises. Dans cette section, nous traitons des relations hiérarchiques qui existent entre ces éléments.

Les données manipulées par l'outil TACTIC se regroupent en trois ensembles principaux :

- Les réseaux : Cet ensemble de données fournit la description complète d'une infrastructure de transport (e.g. noeuds et liens du réseau physique) dans une région, ainsi qu'un réseau de service initial opérant sur ce réseau physique. Ces données comprennent possiblement des résultats provenant soit d'un algorithme de recherche opérationnelle où d'une situation déjà existante. Les résultats sont les itinéraires et les volumes associés par lesquels se fait l'acheminement du fret. Cela spécifie aussi les fréquences finales des services, le flux de marchandises sur les services, les coûts d'opération du système ainsi que le niveau d'activité aux terminus et sur les liens. Cette configuration possible du système de transport est appelée un **scénario.** C'est par l'intermédiaire des scénarios que le planificateur analyse les diverses hypothèses concernant l'état du système de transport. Mentionnons que l'outil TACTIC doit manipuler plus qu'un scénario pour une application.

- La demande de transport : Cet ensemble regroupe toutes les données concernant la demande de transport multiproduits. On y trouve le noeud origine, le noeud destination, la commodité, la demande en tonnes ou en nombre de véhicules ainsi que des coûts d'opération et d'autres valeurs spécifiant le niveau de service désiré pour satisfaire les marchés. Puisque l'utilisateur désire tester plusieurs hypothèses concernant l'état de la demande de transport, nous conservons plusieurs ensembles de ces données.

- Les fonctions : Un aspect important de notre modèle de planification est sa capacité de s'adapter à plusieurs contextes. Ainsi, le planificateur a la possibilité de définir ses propres fonctions de coût et de délai. Avec l'outil TACTIC, cela peut se faire à l'aide d'une table de fonctions déjà prédéfinies ou à l'aide d'expressions algébriques.

En plus de ces trois parties principales, l'outil TACTIC permet à l'utilisateur de spécifier certaines données auxiliaires. Elles constituent des moyens pour améliorer l'interprétation des dessins (e.g. démarcations géographiques, contours politiques, etc.) et elles facilitent l'utilisation de l'outil (e.g. fenêtres prédéfinies, attributs d'affichage tels la couleur et la taille des symboles utilisés, etc.). Elles contiennent aussi des statistiques et autres résultats.

Ces éléments possèdent des caractéristiques qui leur sont propres et forment une hiérarchie qu'il est nécessaire de respecter lors de l'entrée et des modifications des données. Ainsi, avant d'éliminer un lien physique, il faut s'assurer qu'aucun service n'emprunte ce lien. La figure 6.1 schématise les relations qui existent entre les éléments.

Soulignons que la nécessité de considérer explicitement l'information associée aux chemins, tant pour la définition du service que pour l'acheminement du fret, contribue grandement à augmenter tant la taille de

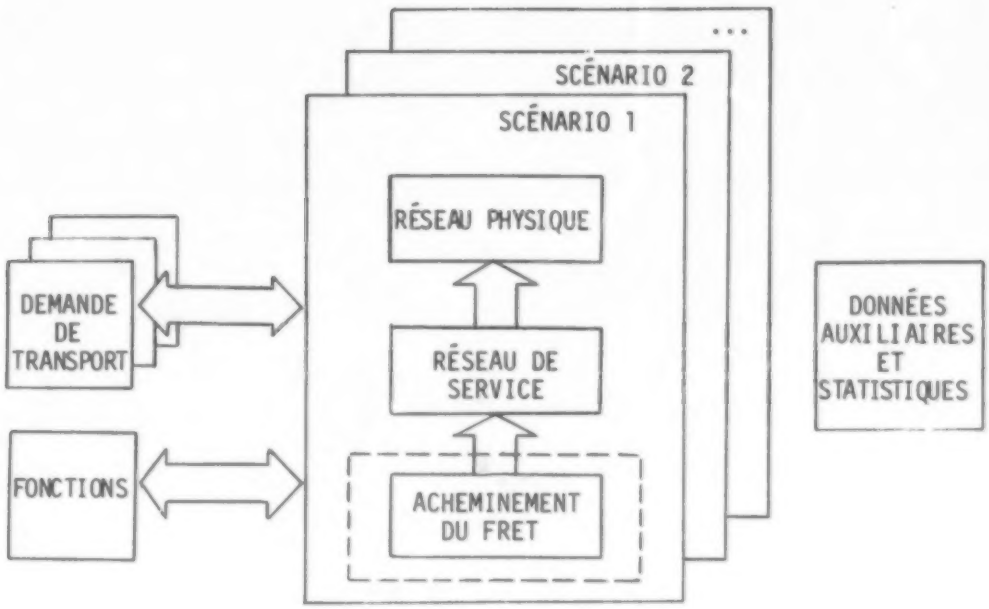


Figure 6.1 : Hiérarchie entre les divers éléments manipulés par l'outil TACTIC

la banque de données que sa complexité. Nous terminons cette section en explorant plus en détail les données associées aux chemins.

Le **réseau physique** est essentiellement composé de noeuds et de liens.

Un **noeud** désigne un endroit précis dans la région géographique considérée. Sa position est donnée par une paire de coordonnées XY. Un noeud possède un identificateur interne (numéro entre 1 et n, où n est le nombre de sommets), un identificateur externe, défini par l'usager, formé d'au plus quatre caractères alphanumériques et un type représentant les jonctions (J), les stations (S) et les terminus (T) du réseau. Des données de l'usager peuvent aussi être spécifiées pour chacun des noeuds.

Un **lien** est une connection directionnelle reliant deux noeuds. Ils représentent soit des liens physique (voies ferrées, routes), soit conceptuels (lignes maritimes, aériennes, etc.). Un lien est défini par un noeud origine et par un noeud destination et il possède un type. Un lien possède aussi des caractéristiques physiques (e.g. distance, capacité, etc.) et des données de l'usager. Les boucles, c'est-à-dire les liens reliant un noeud à lui-même et les liens parallèles, reliant les mêmes deux noeuds dans la même direction, ne sont pas admis.

Sur le réseau physique, se surimpose un **réseau de service** assurant le transport du fret entre les noeuds du réseau. Un service possède un numéro interne et un identificateur (externe), un type spécifiant le mode, la vitesse, la priorité, etc., et une fréquence. Conceptuellement, un service se décompose en **arcs de service,** un arc de service représentant le sous-chemin entre deux arrêts successifs sur la route du service. Un arc de service est défini par une séquence d'arc physiques. Dans ce réseau, les arcs de service parallèles sont permis.

Enfin, la notion d'**itinéraire** est introduite afin de décrire un cheminement possible pour le traffic d'un marché donné. Conceptuellement, un itinéraire se décompose en **arcs d'itinéraire.** Un arc d'itinéraire est défini par une séquence d'arcs de service représentant le sous-chemin entre deux terminus de classification. Mentionnons que les arcs d'itinéraires parallèles sont aussi admis dans ce réseau.

Pour la manipulation des chemins, nous utilisons un modèle de structure en cascade à trois niveaux, illustrée à la figure 6.2, qui permet une économie d'espace. Les flèches de la figure 6.2 indiquent les pointeurs utilisés pour accéder aux différents éléments et montrent aussi la hiérarchie entre ceux-ci. La prochaine section contient une discussion plus approfondie sur l'utilisation de ce schéma par les différents programmes d'affichage et d'optimisation.

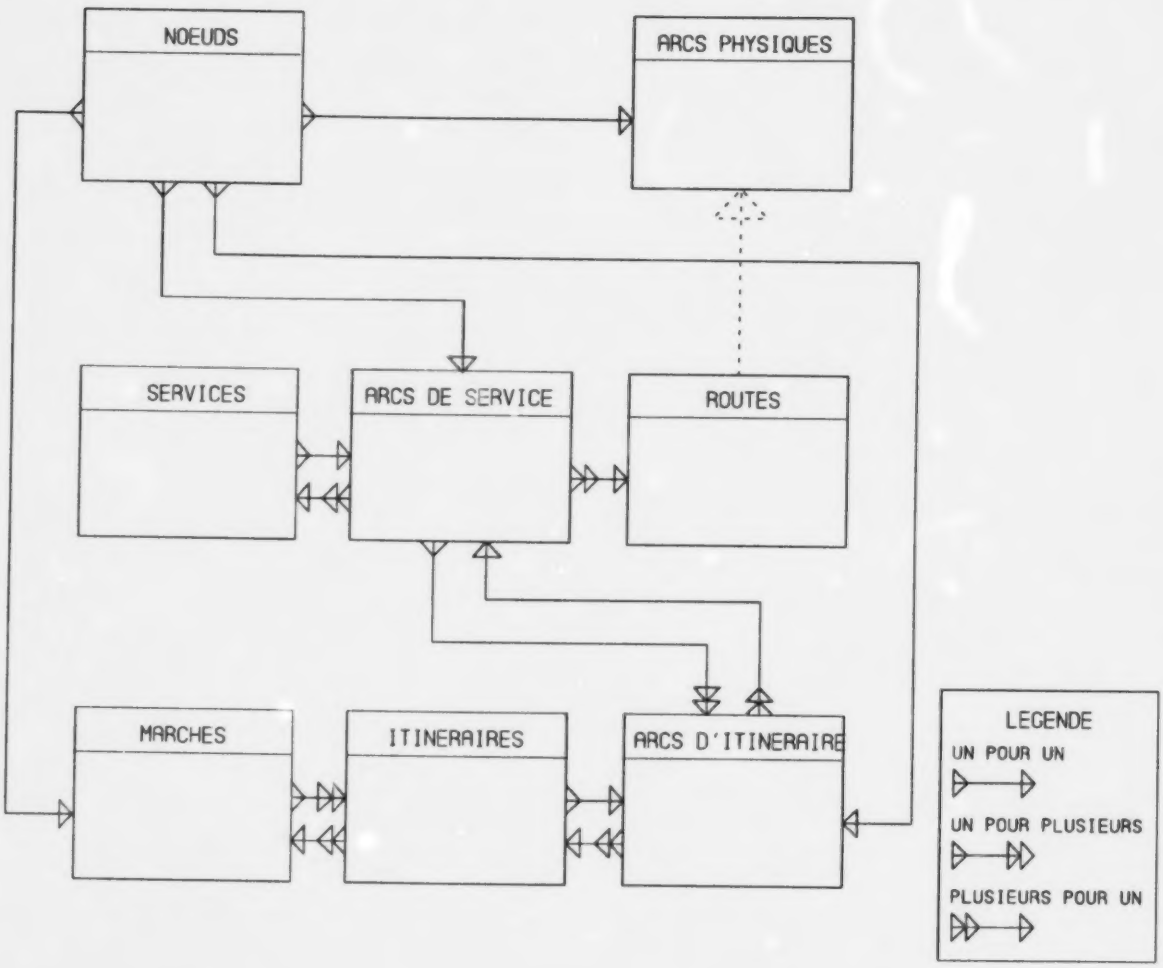


Figure 6.2 : Structure en cascade représentant la hiérarchie entre les réseaux

6.2 Structure de données utilisées par les programmes

Comme nous l'avons mentionné précédemment, l'énorme quantité des données et leur forte interrelation nécessite l'utilisation de structures de données compactes et efficaces tant pour les programmes d'affichage que pour les programmes d'optimisation. Dans cette section, nous traitons de quelques structures de données employées pour la représentation des chemins dans la mémoire centrale de l'ordinateur. Les séquences de programme présentées dans cette section sont écrites en langage C.

Le **réseau physique** est représenté par des listes d'adjacence ("forward star form", voir chapitre 4). La figure 6.3 montre un réseau et sa représentation par des listes d'adjacence dans laquelle nous avons utilisé des listes simplement chaînées.

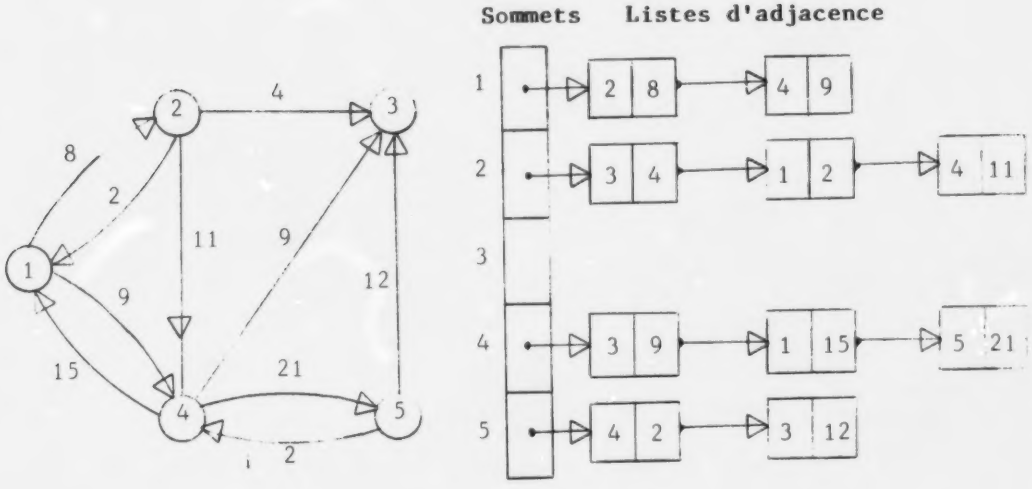


Figure 6.3 : Un réseau et sa représentation par listes d'adjacence

Du point de vue informatique, nous utilisons deux tableaux de structures, **node []** et **link[]**, pour stocker les listes d'adjacence (voir chapitre 4) :

```
struct NODE
{
 struct LINK *a;
 float x,y;
 int i;
 ...
 /*attributs des noeuds*/
}node [NNODE+2];

struct LINK
{
 struct NODE *nd;
 float lngth;
 ...
 /*attributs des liens*/
}link [NLINK+1];
```

Les dimensions respectives de ces deux tableaux sont NNODE+2 et NLINK+1 où NNODE est le nombre de sommets et NLINK est le nombre de liens. Rappelons qu'en langage C les indices des tableaux commencent à zéro. Cette case est perdue car nous numérotons les noeuds 1,...,NNODE. Pour chaque sommet i, la liste des successeurs de i est contenue dans le vecteur **link[]** à partir de l'adresse **node[i].a** jusqu'à l'adresse **(node[i+1].a)-1.** Lorsque la liste des successeurs d'un sommet est vide, nous posons **node[i].a = node[i+1].a.** Ainsi, un parcours du réseau est exécuté par la séquence simple

```
{
 int i;
 struct NODE *j;
 struct LINK *start, *end, *k;
```

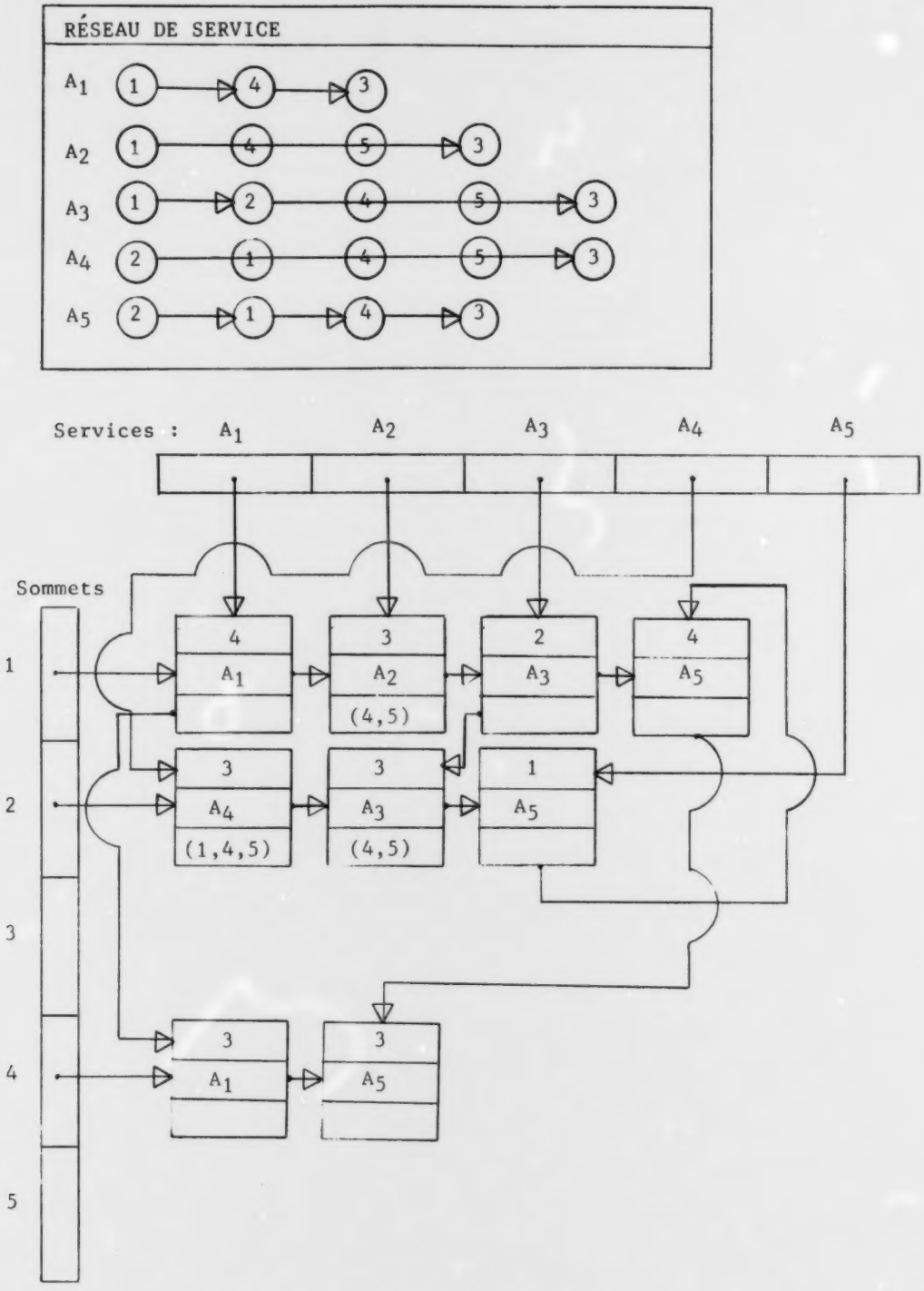


Figure 6.4 : Représentation par multiliste du réseau de service (se référer au réseau physique de la figure 6.3)

```
        for (i=1; i <= NNODE; i ++)
       {
        start = node[i].a;
        end = node[i+1].a;
        for (k=start; k < end; k ++)
          {
           j = k + nd;
           ...
           /*traitement du lien k*/
           ...
          }
       }
    }
```

Le **réseau de service** est aussi emmagasiné sous la forme de liste d'adjacence afin de permettre un accès rapide à tous les arcs de service issus d'un sommet i. Cependant, nous devons ajouter à cette représentation une structure de liste simplement chaînée afin de parcourir dans l'ordre les arcs de service empruntés par chaque service, depuis son sommet origine jusqu'à son sommet destination. Enfin, pour chaque arc de service, il y a un pointeur vers la liste des liens du réseau physique, afin de suivre le chemin physique suivi par les arcs de service. La figure 6.4 illustre cette structure particulière de multiliste.

Nous implantons cette structure de données à l'aide des tableaux de structure **serv[]** et **arc[]** suivants :

```
struct SERV
{
 struct NODE *o, *d;        /*origine et destination du service*/
 struct ARC *first;         /*premier arc du service*/
 ...
 /*attributs des services*/
 ...
}serv [NSERV+1];            /*NSERV = le nombre de services*/
```

```
struct ARC
{
 struct NODE *nd;          /*noeud destination de l'arc de service*/
 struct ROUTE *r;          /*pointeur vers le sous-chemin dans le
                             réseau physique, voir plus loin*/
 int nlink;                /*nombre de liens physiques sur la route,
                             voir plus loin*/
 struct ARC *next;         /*prochain arc de ce même service*/
 struct SERV *sv;          /*service auquel appartient cet arc de
                             service*/
 ...
 /*attributs des services*/
 ...
}arc [NARC+2];             /*NARC = le nombre d'arcs de service*/
```

Aussi, nous ajoutons le champ

```
struct ARC *a0;
```

à la structure **NODE** définie précédemment, afin d'accéder aux successeurs des sommets lorsque nous considérons les arcs de service. Ainsi, la liste de tous les arcs du réseau de service est obtenue par la séquence simple

```
{
 int i;
 struct NODE *j;
 struct LINK *start, *end, *k;

 for (i=1; i <= NNODE; i ++)
{
 start = node[i].a0;
 end = node[i+1].a0;
 for (k=start; k < end; k ++)
   {
    j = k + nd;
    ...
    /*traitement de l'arc de service k*/
    ...
   }
}
}
```

Dans le cas où nous voulons parcourir les services un à un, nous exécutons la séquence suivante :

```
{
 int l;
 struct ARC *k;

 for (l=1; l <= NSERV; l ++)
{
 ...
 /*traitement du service l*/
 ...
 k = serv[l]·first;
 while (k)
   {
    ...
    /*traitement de l'arc k du service £*/
    ...
    k=k → next;
   }
}
}
```

Nous avons observé qu'un très grand nombre d'arcs de service provenant de services différents empruntent la même séquence d'arcs physiques. Dans le but d'économiser l'espace, nous ne répétons pas les séquences identiques. À la place, nous utilisons la structure de données illustrée à la figure 6.5. Afin de décrire les routes physiques empruntées par les arcs de service, nous employons un tableau de structure **route[]** défini ci-bas :

```
struct ROUTE
{
 struct LINK *l;
}route [NROUTE];  /*nombre de liens total dans les routes*/
```

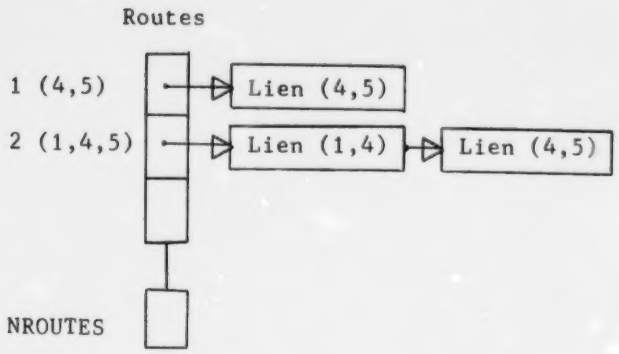


Figure 6.5 : Représentation des routes

Ainsi, nous pouvons accéder à la route physique employée par un arc de service k (où k est un pointeur) de la manière suivante :

```
{
 int n;
 struct ROUTE *ptr;
 struct LINK *l;

 n = k → nlink;
 ptr = k → r;
 while (n)
   {
    l = ptr → l;
    ...
    /*traitement du lien l*/
    ...
    n --;
    ptr ++;
   }
}
```

Finalement, nous employons la structure de multiliste, illustrée à la figure 6.6, pour décrire les **itinéraires.** Mentionnons que les itinéraires sont des entités qui dépendent des marchés. Pour cette raison, nous ne pouvons pas représenter les itinéraires sans les marchés.

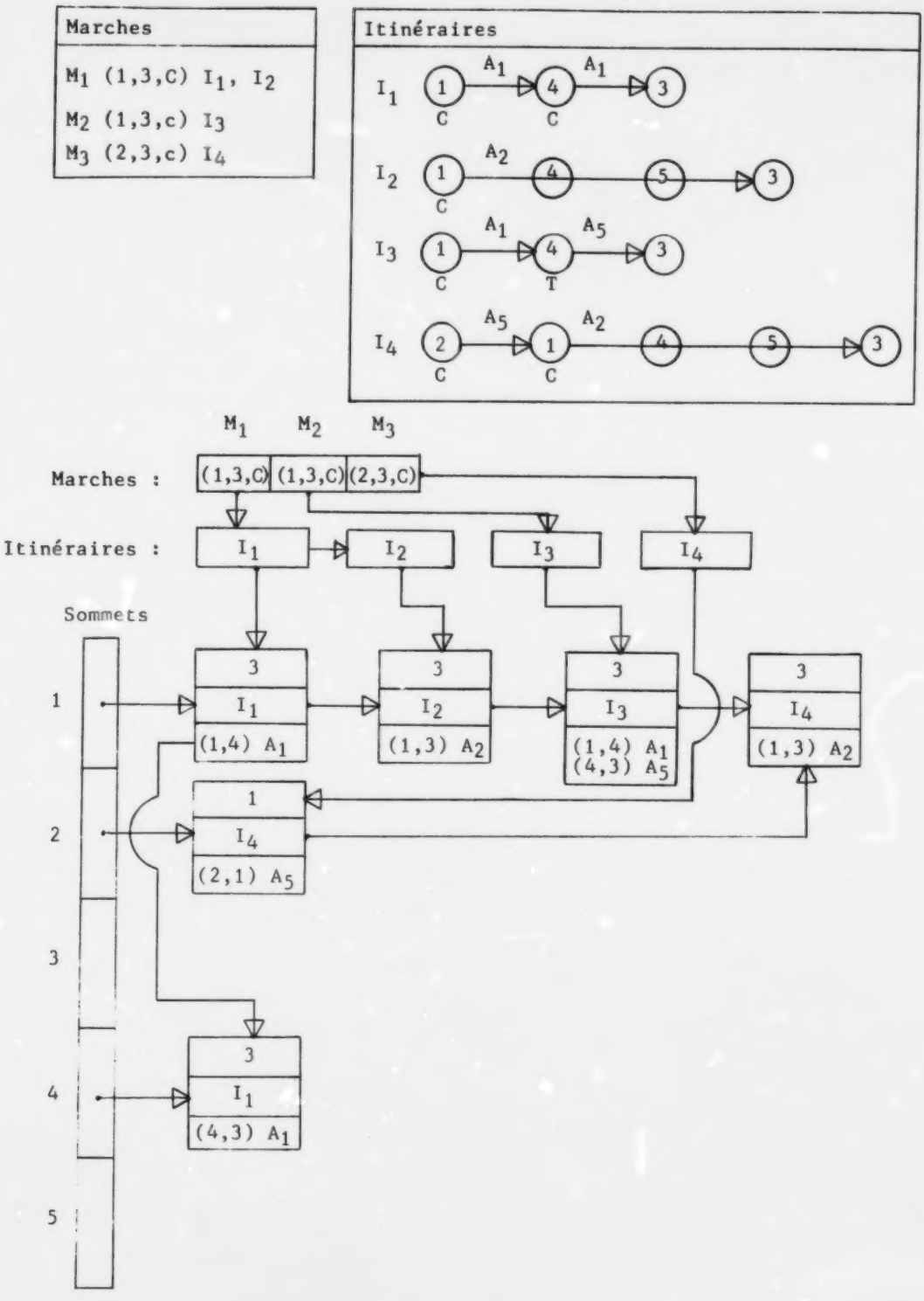


Figure 6.6 : Structure de multiliste pour représenter les itinéraires

Cette structure de données est implantée à l'aide de quatre tableaux de structure, **tclass[], itin[], iarc[]** et **path[],** que nous définissons de la manière suivante :

```
struct TCLASS
{
 struct NODE *o, *d;        /*origine et destination du marché*/
 int c;                     /*commodité*/
 float g;                   /*demande en tonne ou en nombre de véhi-
                              cules*/
 struct ITIN *first;        /*premier itinéraire de ce marché*/
 int nitin;                 /*nombre d'itinéraires pour ce marché*/
 ...
 /*attributs des marchés*/
 ...
}tclass [NTCLASS];          /*où NTCLASS est le nombre de marchés*/

struct ITIN
{
 struct TCLASS *tc;         /*marché que cet itinéraire dessert*/
 struct IARC *first;        /*premier arc d'itinéraire*/
 ...
 /*attributs des itinéraires*/
 ...
}itin [NITIN];              /*où NITIN est le nombre d'itinéraires*/

struct IARC
{
 struct NODE *nd;           /*noeud destination de l'arc d'itiné-
                              raire*/
 struct PATH *p;            /*pointeur vers le chemin dans le réseau
                              de service*/
 int narc;                  /*nombre d'arcs de service sur le che-
                              min*/
 struct IARC *next;         /*prochain arc d'itinéraire de ce même
                              itinéraire*/
 struct ITIN *it;           /*itinéraire auquel appartient cet arc
                              d'itinéraire*/
 ...
```

```
 /*attributs des arcs d'itinéraire*/
 ...
}iarc [NIARC];          /*où NIARC est le nombre total d'arcs
                          d'itinéraire*/

struct PATH
{
 struct ARC *a;         /*arc de service*/
}path [NPATH];          /*nombre d'arcs de service total dans les
                          chemins*/
```

De plus, nous ajoutons le champ

```
struct IARC *a1;
```

à la structure **NODE** définie précédemment afin d'accéder aux successeurs des sommets dans le réseau des itinéraires. Ainsi, la liste de tous les arcs d'itinéraire est obtenue par la séquence simple

```
{
 int i;
 struct NODE *j;
 struct IARC *start, *end, *k;

 for (i=1; i <= NNODE; i ++)
{
 start = node[i].a1;
 end = node[i+1].a1;
 for (k=start; k < end; k ++)
   {
    j = k + nd;
    ...
    /*traitement de l'arc d'itinéraire k*/
    ...
   }
}
}
```

Dans le cas où nous voulons parcourir les itinéraires un à un, nous exécutons la séquence suivante :

```
{
 int l;
 struct IARC *k;

 for (l=1; l <= NITIN; l ++)
 {
 ...
 /*traitement de l'itinéraire l*/
 ...
 k = itin[l].first;
 while (k)
   {
    ...
    /*traitement de l'arc k de l'itinéraire l*/
    ...
    k = k -> next;
   }
 }
}
```

Enfin, nous suivons les chemins, dans le réseau de service, empruntés par les arcs d'itinéraire de la manière suivante :

```
{
 int n;
 struct PATH *ptr;
 struct ARC *a;

 n = k -> narc;          /*k est un pointeur sur un arc d'itinéraire,
                           qui doit être défini au prélable*/
 ptr = k -> p;
 while (n)
   {
    a = ptr -> a;
    ...
    /*traitement de l'arc de service a*/
    ...
    n --
    ptr ++;
   }
}
```

Avant de passer à la prochaine section, nous désirons faire quelques remarques importantes concernant le travail que nous venons de présenter.

En premier lieu, nous avons constaté que certains auteurs (e.g. Person, 1987) suggèrent l'emploi systématique de structures faisant référence à elle-même, ou récursives pour implanter des listes. La figure 6.7 illustre une liste simplement chaînée implantée avec une structure récursive. Cette implantation "type PASCAL" est très élégante. Cependant, l'utilisation d'un pointeur sur l'élément suivant représente un gaspillage de la mémoire centrale quand les réseaux ne sont pas modifiés dans un programme (voir Aho et al., 1987). Pour cette raison, nous avons évité l'usage de cette méthode pour les programmes d'affichage graphique.

```
/*noeud d'une liste simplement chaînée*/
 struct liste
   {
    ...
    /*éléments de la liste*/
    ...
    struct liste * suivant;
   }
```

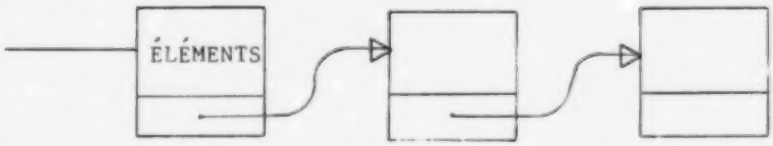


Figure 6.7 : Une liste chaînée simple

Mentionnons que nous n'avons pas encore choisi le type d'implantation ni défini les règles de modification des éléments des différents niveaux de réseaux. C'est une étape ultérieure du projet. Pour cette raison, nous n'avons pas encore déterminé les structures de données pour effectuer ces opérations. Cependant, une solution possible consiste à employer les tableaux séquentiels que nous venons de décrire. Cela dépend aussi de l'organisation des données sur disque. La prochaine section contient une discussion sur le problème difficile de la manipulation des données.

Soulignons que ces structures de données se ramènent à l'utilisation de tables séquentielles efficacement implantées en langage C car nous avons utilisé une caractéristique de ce langage (qu'on ne retrouve pas dans le langage PASCAL), qui permet aux pointeurs de contenir l'adresse d'une case d'un tableau (voir chapitre 4). Jusqu'à maintenant, les éléments dans ces tables séquentielles ne sont pas triés selon leur numéro interne. Cela dépend de la structure de données employée pour conserver ces éléments dans la banque de données. Nous discutons de cet aspect à la fin de la prochaine section. Mentionnons que ces structures de données se sont avérées très efficaces et pratiques lors du développement des modules graphiques décrits au chapitre 7.

Dans notre description, nous avons omis volontairement d'autres listes, moins importantes, dans le but de ne pas alourdir cette longue présentation. Notamment, pour chaque lien du réseau physique, nous avons un pointeur vers la liste des arcs de service qui empruntent ce lien. Aussi, pour chaque arc de service, nous avons un pointeur sur une liste des arcs d'itinéraires qui empruntent cet arc. Actuellement, ces deux listes sont aussi implantées avec des pointeurs sur des tableaux.

6.3 Banque de données

Présentement, l'exécution des programmes du noyau méthodologique consiste à lire plusieurs grands fichiers séquentiels de données dans le but de traiter un problème spécifique. Les fichiers contiennent les données du problème et certaines instructions sous forme de paramètres. Ils sont préparés à l'avance par l'usager. À la fin de l'exécution, le programme génère les résultats en forme de fichiers de sorties et/ou de rapports imprimés. Puisque les opérations sont arrangées en un ordre fixe, qui est établi à l'avance, les données d'entrée sont disposées dans l'ordre où elles sont utilisées. Ce type de traitement des données porte le nom de traitement par lot.

Un système interactif-graphique ne peut pas anticiper l'ordre dans lequel se déroulent les différentes tâches. Pour cette raison, la lecture séquentielle de toutes les données requises pour une opération est hors de question. Aussi, n'oublions pas la quantité énorme de résultats générées par le modèle.

Nous remarquons donc la nécessité d'organiser les données afin de permettre un accès rapide et sélectif aux éléments requis pour accomplir une tâche. Cette organisation interne des données est faite sous la forme d'une banque de données, c'est-à-dire un fichier-disque à accès direct (ou aléatoire), qui contient les éléments, leurs attributs et leurs relations, et qui permet un échange d'informations efficace et sélectif. Mentionnons que ce fichier est dans une forme binaire. Cela présente l'avantage de réduire considérablement l'espace disque nécessaire, en plus d'accélérer les opérations de lecture et d'écriture. En langage C, les fichiers binaires sont manipulés avec les fonctions **write()**, **read()** et **lseek()**, cette dernière permettant les accès aléatoires.

Actuellement, nous n'avons pas encore implanté la banque de données. C'est la prochaine étape du projet. Pour l'instant, les programmes graphiques utilisent des fichiers séquentiels d'entrées. Cependant, nous avons étudié quelques structures de banque de données utilisés dans d'autres systèmes interactifs tels EMME/2 (Spiess, 1984), PLANAIR (Lebeuf, 1985) et STAN (Crainic, 1989). Considérant que les étapes d'utilisation sont, en général, similaires nous avons décidé d'adopter les trois principes, simples, de fonctionnement suivant :

1) Organisation verticale des données;
2) Données de base et résultats emmagasinés dans la banque de données;
3) Détection des erreurs à l'entrée.

Reprenons un à un ces concepts :

1) Organisation verticale des données

L'organisation **verticale** des données consiste à utiliser un enregistrement pour chaque attribut qui contient la valeur de cet attribut pour chaque élément. Contrairement à l'organisation horizontale, qui consiste à accumuler dans un enregistrement les valeurs de tous les attributs associées à un élément, l'organisation verticale permet de ne lire que les attributs nécessaires à un programme. De ce fait, la méthode d'organisation verticale réduit le nombre d'accès-disque tout en assurant une meilleure gestion de l'espace mémoire centrale.

2) Données de base et résultats stockés dans la banque

TACTIC constitue un environnement de travail interactif-graphique pour assister un planificateur dans sa tâche d'analyse et de conception d'un plan de transport. De plus, avec un tel outil, le planificateur peut manipuler une solution afin qu'elle respecte les règles et les critères non inclus dans le modèle. Dans ce contexte, les résultats d'un algorithme de

recherche opérationnelle sont aussi les données de base pour une étape subséquente d'analyse ou de manipulation. Donc, il est clair que les données de base et les résultats doivent être emmagasinés comme une partie intégrante de la banque de données. Mentionnons que cette manière de procéder contribue grandement à augmenter tant la taille de la banque de données que sa complexité (e.g. 4 000 à 5 000 itinéraires pour une seule étude de planification), mais à notre avis c'est le seul moyen de mettre en oeuvre un véritable outil de travail interactif-graphique.

3) Détection des erreurs à l'entrée

Un problème important concernant la gestion d'une banque de données est l'intégrité des données qui sont contenues. Une façon simple et élégante de solutionner ce problème est d'exiger que la banque soit toujours dans un état consistant. Pour ce faire, toute modification apportée à la banque de données n'est effectuée qu'après avoir vérifié la validité de l'opération et s'être assuré qu'elle ne cause aucune inconsistance. Définissant l'état initial vide comme un état consistant, la règle de détection des erreurs à l'entrée garantit que les transitions se font d'un état consistant à un autre. Ainsi, un programme qui utilise la banque de données est assuré de la trouver en bon état et ce même programme a la responsabilité de veiller à ce que chaque modification qu'il apporte soit permise, de façon à conserver la banque dans un état consistant. Mentionnons que les modifications apportées à la banque sont des opérations simples : addition, suppression et modification d'un élément. De plus, il suffit de tester, pour chaque opération, si les valeurs des attributs sont valides et si les relations entre l'élément modifié et les autres sont consistantes. À cet effet, les considérations concernant la hiérarchie des éléments que nous avons mentionnées à la section précédente sont importantes.

Mentionnons que la banque de données de EMME/2 (Spiess, 1984) est aussi basée sur les deux concepts suivants :

4) Absence de redondance dans les données de la banque;

5) Les données sont stockées dans des tables séquentielles, triées selon les numéros internes des éléments.

Cependant, nous remarquons des différences entre les données que nous manipulons et celles gérées par EMME/2 (Spiess, 1984), les principales étant celles-ci :

- Réseau de service multimode avec une représentation chemin;
- Acheminement des marchandises par des itinéraires (chemins dans le réseau de service);
- Flots aux noeuds sous forme de liste;
- Flots sur des chemins.

Dans notre contexte, les concepts d'absence de redondance et d'utilisation de tables séquentielles sont difficilement applicables, malgré leurs nombreux avantages. Passons en revue les points 4 et 5 ci-haut mentionnés.

En premier lieu, lorsqu'une banque de données est modifiée fréquemment de façon interactive, l'**absence de redondance** est une méthode efficace pour s'assurer de la consistance des informations qui y sont contenues. De plus, si une politique de détection des erreurs à l'entrée est appliquée, l'absence de redondance allège le travail nécessaire pour valider et effectuer les modifications. Le désavantage associé à cette méthode est un effort supplémentaire de calcul pour regénérer ces données chaque fois que nous en avons besoin. Pour certaines applications (EMME/2 : Spiess, 1984; PLANAIR : Lebeuf 1985; STAN : Crainic, 1989), ce surplus de travail est mineur et chaque vecteur de données secondaires est recalculé chaque fois qu'il est utilisé. Dans TACTIC, cela n'est pas toujours désirable à cause des différents niveaux de réseaux. Prenons l'exemple de

l'affichage du flux total de marchandises sur les liens du réseau. Pour chaque lien, nous devons d'abord consulter la liste de tous les arcs de service qui empruntent ce lien. Ensuite, pour chacun de ces arcs nous consultons tous les itinéraires qui y passent pour finalement calculer le flux total. Cela peut causer un délai inadmissible surtout si, à cause de leur taille, la description des services et des itinéraires ne peuvent résider dans la mémoire centrale de l'ordinateur. Dans un tel cas, il semble préférable de conserver cette valeur redondante.

Le problème se ramène donc à différencier les données de base et les données redondantes. La définition la plus simple de ce concept consiste à dire que toutes les données qui peuvent être calculées ou déduites sont des données redondantes. Dans ce cas, la liste même des arcs de service qui empruntent un lien devient une information redondante. Or, l'obtention de cette liste nécessite un travail considérable.

Une solution envisageable est de ne pas conserver dans la banque de données les éléments redondants mais d'utiliser un espace mémoire auxiliaire qui contiendrait ces éléments redondants. Cet espace mémoire peut être, soit un fichier sur disque, soit une partie de la mémoire centrale de l'ordinateur qui est initialisé au début de chacun des programmes. Malheureusement, cette solution possible ne règle pas tous les problèmes. En effet, lors des modifications des éléments de la banque, cela peut requérir plus de travail de validation et de calcul pour maintenir la banque dans un état consistant. Donc, il s'agit pour nous d'évaluer les compromis à faire entre les avantages et les désavantages liés à l'emmagasinement ou non d'une donnée redondante. Cette analyse doit se faire, cas par cas et nous continuons de travailler sur ce sujet.

Le second point concerne l'ordre dans lequel les éléments sont disposés dans la banque de données. La façon la plus simple consiste à utiliser des tables séquentielles ordonnées, triées selon les numéros

internes des éléments. Cette structure de données comporte plusieurs avantages :

- Une recherche dichotomique peut être directement utilisée pour accéder à un élément particulier.
- Deux structures qui contiennent les mêmes éléments sont physiquement identiques, peu importe l'histoire de leur développement.
- Il est facile d'utiliser une lecture par tampon lorsque l'espace mémoire central est limité.
- Cette organisation permet de comparer facilement deux structures complexes en les parcourant en parallèle.

Malheureusement, cette méthode augmente le nombre d'accès disque nécessaires pour effectuer une modification. Ainsi, à chaque fois qu'un élément est ajouté ou retranché, plusieurs autres doivent être décalés. La figure 6.8 montre un petit exemple illustrant le problème.

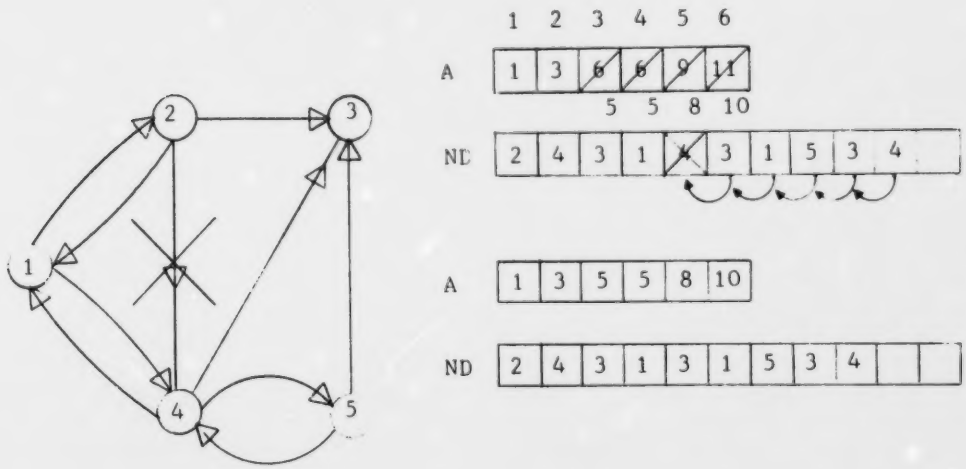


Figure 6.8 : Exemple de l'élimination du lien (2,4) quand le réseau est stocké dans des tables séquentielles

Une solution pour pallier à ces inconvénients consiste à utiliser des listes chaînées. Dans ce cas, les modifications à la banque de données sont triviales (voir Aho et al., 1987). Toutefois, cette méthode possède aussi des désavantages, le plus considérable étant l'espace mémoire supplémentaire requis pour conserver les faux-pointeurs ou curseurs. Cet aspect relié à l'utilisation des listes chaînées peut induire un problème encore plus considérable que le temps nécessaire pour la mise à jour des tables séquentielles.

Comme précédemment, nous n'avons pas solutionné ce problème et nous poursuivons nos travaux en comparant les désavantages et les utilités liés à ces deux approches (tables séquentielles versus listes chaînées). De plus, nous devons tenir compte des développements technologiques dans le domaine des disques de stockage et de l'utilisation des antémémoires ("cache memory", voir Lebeuf, 1985).

6.4 Conclusions

Dans ce chapitre, nous avons présenté les hiérarchies qui existent entre les différents éléments que l'outil TACTIC doit manipuler. Nous avons montré la structure relativement complexe pour la représentation des différents niveaux de réseau. Nous avons utilisé des caractéristiques du langage C pour implanter des structures de données efficaces, pour les programmes d'affichage graphique, et qui prennent peu d'espace mémoire. Cependant, les problèmes de l'organisation des données pour les modifications et l'emmagasinement disque n'ont pu être solutionnés jusqu'à maintenant.

Mentionnons que la dimension "chemin" est particulière à l'outil TACTIC. Pour cette raison nous ne pouvons pas facilement adapter des principes de développement des banques de données employés dans d'autres systèmes interactifs-graphiques adaptés aux problèmes de réseau tels EMME/2 (Spiess, 1984) et STAN (Florian et al., 1987). C'est notamment le cas pour le principe d'absence de redondance dans la banque.

Finalement, nous remarquons que le problème de l'organisation des informations dans la banque est semblable au problème du choix de la structure de données pour les programmes qui effectuent des modifications interactives graphiques, et pour les programmes d'optimisation (génération des itinéraires). Jusqu'à maintenant, nous avons envisagé deux solutions : les tables séquentielles et les listes chaînées. Nous poursuivons nos travaux en comparant les avantages et les désavantages. De plus, nous comptons faire des lectures plus poussées sur ce sujet.

CHAPITRE 7

INTERFACE GRAPHIQUE

Notre mécanisme oeil-cerveau de reconnaissance de formes nous permet d'analyser rapidement plusieurs types de données lorsqu'elles sont présentées graphiquement.

L'informatique graphique est la création, l'emmagasinement et la manipulation de modèles d'objets ainsi que leur représentation graphique à l'aide d'un ordinateur. Un cas particulier est l'informatique interactive graphique où l'usager contrôle dynamiquement les images générées. Un environnement de travail est créé et une conversation est établie entre l'ordinateur et l'utilisateur via des menus, icônes et dispositifs d'entrée tel le clavier, la souris, la tablette graphique et autres.

L'apparition des périphériques graphiques performants (moniteur couleur, imprimante à points et au laser, etc.) à des prix abordables ont permis la diffusion des systèmes graphiques. Il n'y a qu'à penser aux jeux et programmes faisant appel au graphisme pour les ordinateurs personnels. Quant aux domaines scientifiques, les systèmes CAO (conception assistée par ordinateur) et DAO (dessin assisté par ordinateur) sont très répandus auprès des ingénieurs et architectes. On trouve une discussion plus élaborée sur les applications de l'informatique interactive-graphique dans Foley et Van Dam (1984).

Du côté de la recherche opérationnelle, la puissance de la représentation graphique des réseaux a été suffisamment illustrée par plusieurs systèmes interactifs-graphiques pour la planification des réseaux de transport urbain de personnes (voir chapitre 1).

Au cours de ce dernier chapitre, nous présentons les principes généraux de définition et de développement des modules interactifs-graphiques de TACTIC. Nous discutons de la mise en oeuvre informatique. Nous introduisons la bibliothèque de fonctions graphiques en langage C mise au point par nous. Par la suite, nous traitons des solutions adoptées jusqu'à maintenant pour l'affichage des éléments du problème de planification. Finalement, nous montrons comment TACTIC se présente du point de vue d'un usager, en donnant un aperçu général des modules développés et de leur fonctionnement. Notre discussion est illustrée à l'aide de nombreuses sorties graphiques.

7.1 Structure d'un module graphique

Dans le but de rendre efficace l'interface entre l'humain et l'outil informatique, il faut que l'usager n'ait qu'à apprendre un petit ensemble de règles simples qui s'appliquent uniformément à tous les modules. Ces règles incluent la génération d'un dessin, son interprétation et toutes les manipulations graphiques subséquentes. Donc, les programmes graphiques doivent se comporter d'une façon consistante. La standardisation des modules possède aussi un autre avantage considérable : cela facilite énormément la conception, la programmation, la documentation et l'extension de l'outil.

Du point de vue informatique, chaque module correspond à un programme autonome. Pour un utilisateur, l'exécution d'un module graphique s'articule autour de cinq étapes illustrées à la figure 7.1;

1. Appel : chargement du module dans la mémoire de l'ordinateur, initialisation de certains paramètres aux valeurs par défaut, lecture de la banque de données;

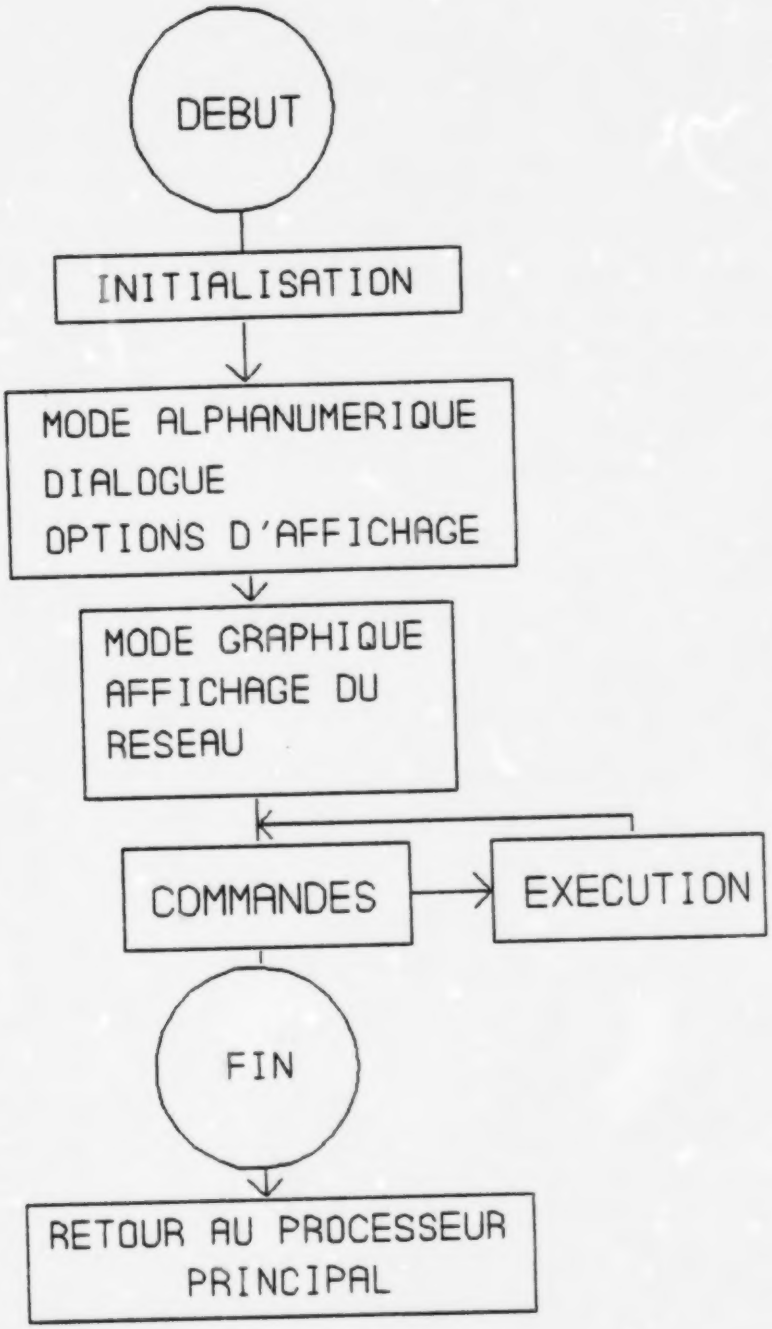


Figure 7.1 : Structure des modules graphiques

2. Dialogue : partie interactive, en mode alphanumérique, "texte", permettant à l'usager de spécifier les paramètres d'exécution du module : fenêtre initiale, sous réseau de travail, type d'affichage, type et désignation de l'unité destinataire des résultats, échelle des attributs, etc.;

3. Affichage : affichage des informations demandées selon les spécifications et sur l'unité indiquée par l'usager;

4. Interaction : partie interactive-graphique permettant à l'utilisateur d'interroger, à l'aide de commandes simples, le contenu de la banque de données. Les informations sont présentées graphiquement;

5. Fin : le module repasse le contrôle au processeur principal.

Notons qu'un terminal graphique n'est essentiel que pour l'étape 4. Quand le travail est fait à l'aide d'un terminal non graphique, cette étape est omise, tandis qu'à l'étape 2, l'unité devant recevoir les résultats de l'étape 3 est, non pas l'écran, mais un fichier qui est disponible pour les travaux ultérieurs. Ce fichier peut être soit un fichier-texte pour les rapports, soit si l'information est graphique, un fichier des dessins ("plot file"). Nous référons le lecteur à la section 7.5.1 pour la description des commandes pour la manipulation du fichier des dessins.

7.2 Les écrans

Le travail en mode graphique se fait à l'aide d'écrans. Un écran est une division en zones de la surface totale d'affichage. Nous avons fixé le ratio d'aspect de la surface totale d'affichage à 4X:3Y. C'est le format trouvé dans la plupart des terminaux graphiques contemporains.

Afin de faciliter l'utilisation du système, chaque zone n'a qu'une seule application possible : titre, légende, dessin de réseau, entrée des commandes, etc. De plus, nous prévoyons utiliser seulement deux types d'écran : le **dessin** ("plot") et la **feuille de travail** ("graphic work sheet").

Ces deux formats sont nécessaires pour tenir compte des deux modes d'utilisation d'un système graphique : le format dessin est utilisé pour toute sortie graphique qui a comme but d'afficher et de permettre d'interroger des éléments contenus dans la banque de données, tandis qu'on se sert de la feuille de travail pour toute modification interactive-graphique du contenu de la banque de données.

À ce stade-ci du projet, les feuilles de travail ne sont pas encore implantées. Jusqu'à maintenant, nous avons concentré nos efforts à surmonter les difficultés d'affichage des éléments du modèle de planification tactique plutôt qu'aux modifications interactives-graphiques de la banque de données. On trouve une discussion élaborée sur les feuilles de travail dans Spiess (1984).

La figure 7.2 est un exemple d'une sortie graphique en format dessin. L'écran de type dessin est divisé en six zones, dénotées A à F dans la figure 7.3. Voici le rôle associé à chacune des zones :

A) En haut de l'écran on trouve toujours le titre du module en cours d'exécution. Cette information est générée par le programme et ne peut pas être modifiée par l'usager.

B) Dans le coin supérieur droit apparaît le logo de TACTIC.

C) La partie principale de l'écran contient l'affichage des éléments de la banque de données. Cette zone couvre 78 % de la surface totale d'affichage.

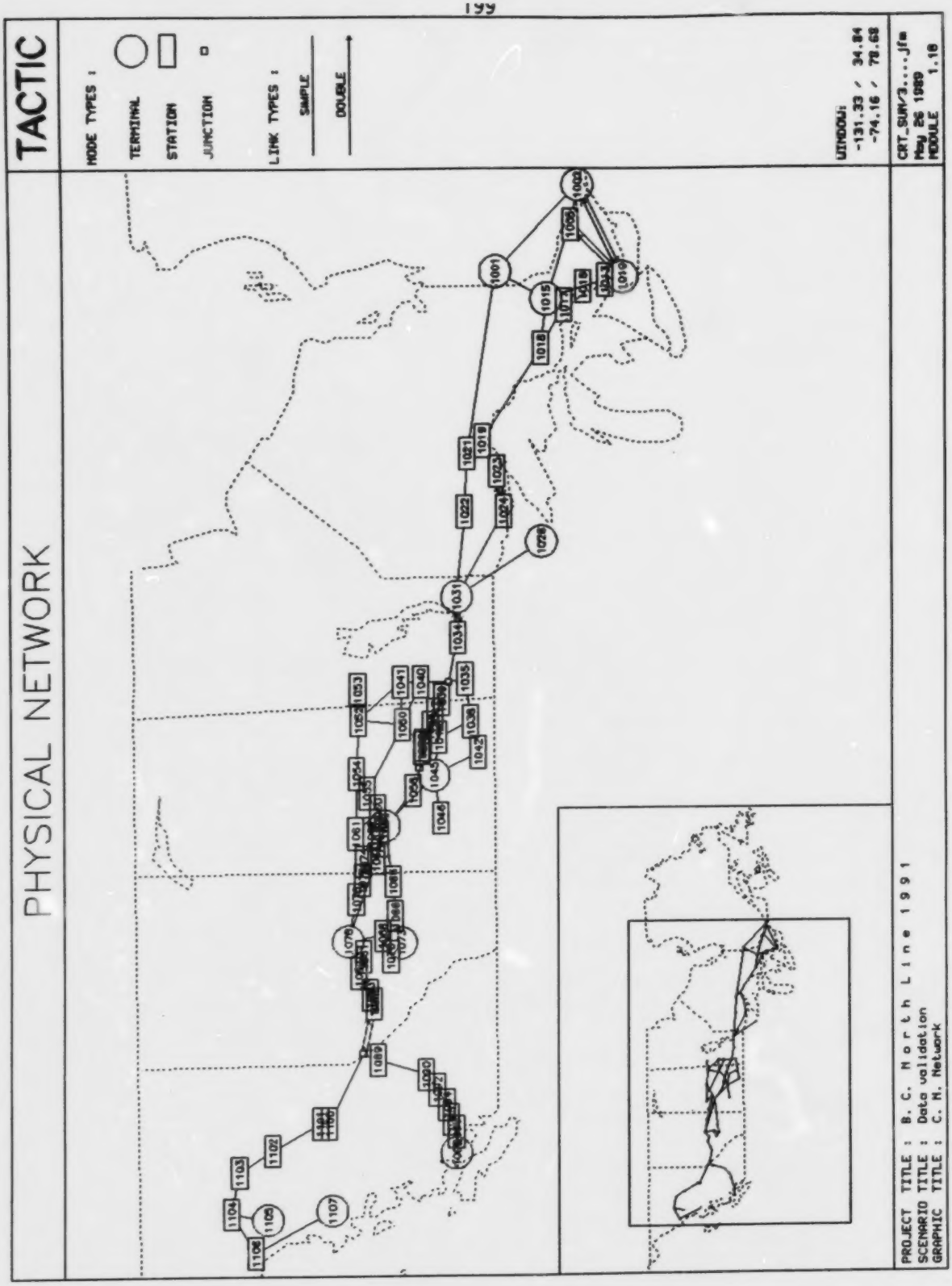


Figure 7.2: Exemple d'une sortie graphique en format dessin

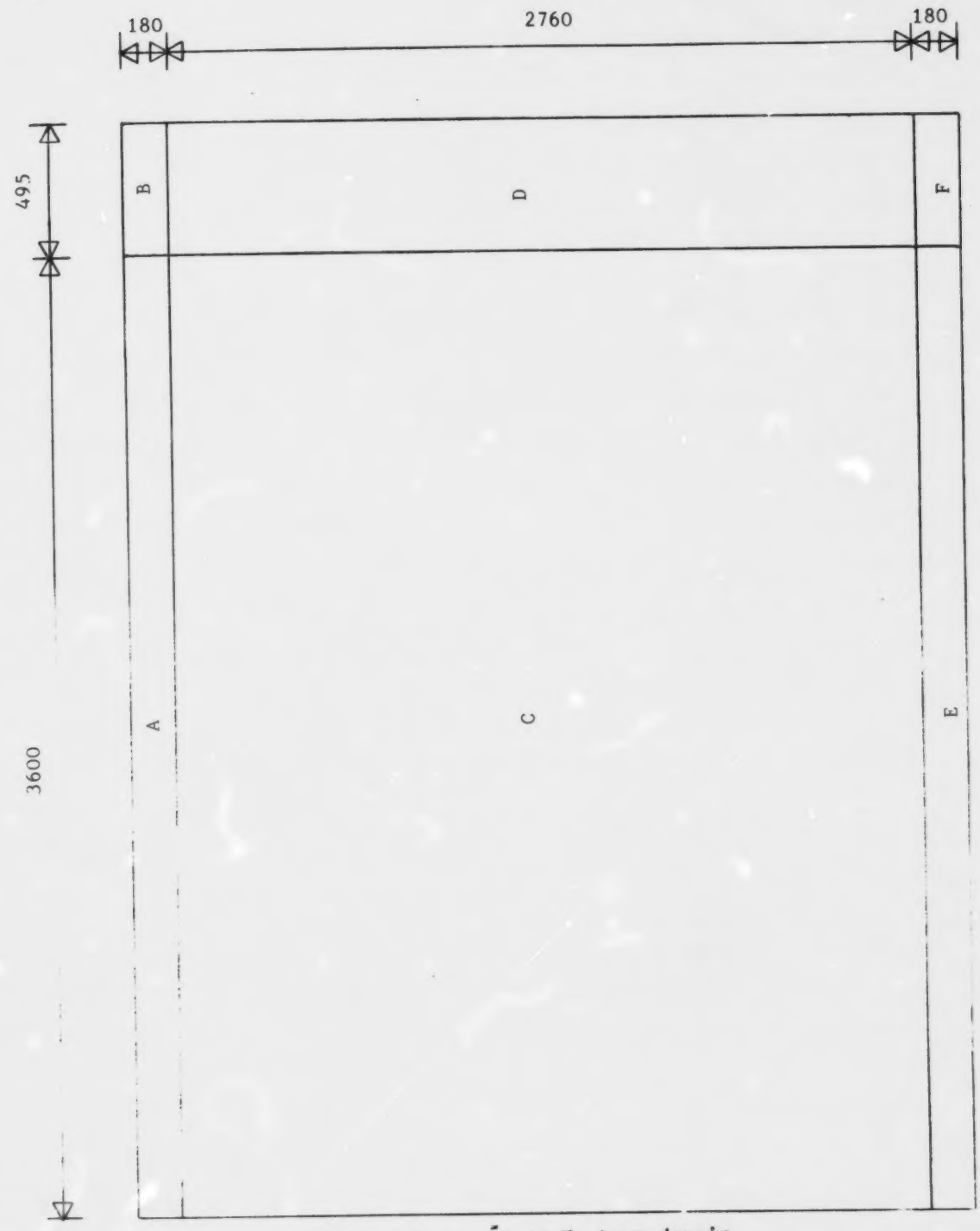


Figure 7.3 : Écran de type dessin

D) La marge de droite contient toute l'information nécessaire à l'interprétation des détails du dessin. On trouve ici la description des symboles utilisés pour représenter les noeuds et les liens du réseau, l'échelle des attributs et les coordonnées qui définissent la portion du réseau dessinée. Cette zone porte le nom de **légende.**

E) En bas de l'écran, à gauche, nous indiquons la provenance des données affichées. La première ligne, commençant par "PROJECT TITLE", permet d'identifier la banque de données utilisée. C'est le titre du projet. Le titre du projet est fixé lors de la création de la banque de données. La ligne du milieu spécifie le scénario de la banque de données sur lequel l'usager travaille. Enfin, la dernière ligne permet à l'usager de personnaliser le dessin par un titre ou un commentaire. Pendant l'exécution d'un module, cette ligne peut être modifiée par une commande graphique (voir section 7.8). L'espace libre à droite de ces descriptions est réservé à l'entrée des commandes graphiques. Ces commandes sont décrites à la section 7.8.

F) Finalement, le coin inférieur droit montre l'identificateur de l'installation, les initiales de l'usager, la date et le numéro du module en cours d'exécution.

Notons que cette composition de l'écran de type dessin est semblable à celle utilisée dans EMME/2 (Spiess, 1984). Nous avons analysé d'autres arrangements (voir Crainic et Mondou, 1986), mais nous croyons que celui que nous adoptons dans ce mémoire convient mieux à nos besoins.

7.3 La question technologique

Actuellement, les programmes du noyau méthodologique se trouvent sur l'ordinateur Cyber 835/855 de l'Université de Montréal. Mentionnons que nous travaillons aussi à la conversion de ces programmes en langage C sur un environnement UNIX (voir section 7.4). Le système graphique est développé sur l'ordinateur Sun 3/180 du Centre de recherche sur les

transports. Cet ordinateur est une machine de catégorie "working station", utilisant un système d'exploitation UNIX. Les deux ordinateurs peuvent facilement communiquer.

Les dessins produits par TACTIC requièrent seulement que l'équipement graphique puisse tracer un vecteur reliant deux points. Bien que le système n'exige pas de résolution minimale en nombre de pixels, une résolution de 640 par 480 pixels (e.g. carte EGA) semble constituer une borne inférieure. De plus, faisant suite aux choix effectués pour les problèmes d'affichage (voir section 7.7), nous avons choisi une implantation des modules graphiques utilisant des terminaux avec écrans à rafraîchissement permettant l'effaçage sélectif de l'écran. Les terminaux à haute résolution de ce type actuellement utilisés au Centre de recherche sur les transports sont :

Tektronix 4115b, terminal polychrome, à haute résolution :
1280 x 1024 pixels;

Visual 550, terminal monochrome avec une résolution de :
768 x 586 pixels, émulateur Tektronix.

Nous prévoyons également une implantation pour des terminaux n'offrant pas cette possibilité, tel le Tektronix 4014, un terminal monochrome à très haute résolution, soit 4096 x 3120 pixels.

Enfin, nous envisageons une mise en oeuvre pour des appareils non compatibles avec Tektronix telles les cartes EGA/VGA (voir section 7.5).

7.4 Le langage de programmation

Nous avons mené nos premières expérimentations graphiques en faisant appel à QPLOT, une bibliothèque de routines graphiques de base écrites en FORTRAN par Heinz Spiess (Spiess, 1984). La programmation du système était également faite en FORTRAN (voir Crainic et Mondou, 1986).

Or, nous remarquons depuis une dizaine d'années que l'équipement informatique nécessaire pour pouvoir appliquer nos modèles de planifications a subi des changements radicaux. Aujourd'hui des micro-ordinateurs

d'une puissance de calcul comparable à des ordinateurs de grande taille ("main frame"), d'il y a quelques années, sont disponibles à une fraction du prix. L'évolution visible dans ce domaine se traduit par la présence de micro-ordinateurs à grande vitesse (e.g. microprocesseur Intel 80386 à 25 Mhz) et à grande mémoire (e.g. 4 mégaoctets ou plus). Ce progrès technologique permet aux petites et aux moyennes entreprises de transport d'employer des méthodes de travail informatisées. Dans ce contexte, nous avons pour objectif d'implanter TACTIC sur des micro-ordinateurs afin de favoriser sa diffusion future.

Parallèlement à ce progrès technologique, nous constatons la popularité grandissante du langage C auprès des concepteurs de programmes informatiques. Cela provient du fait que le langage C facilite la conception et l'application de projets de programmation complexes. Cette caractéristique est largement due à l'efficacité de son code (voir chapitre 4), la concision des énoncés, le contrôle sur la représentation interne des données et leur durée de vie en mémoire ainsi que la grande souplesse que procure l'usage correct des fonctions. Nous soulignons que le C est un langage très "portable" et ses fonctions peuvent être appelées à l'intérieur d'un programme écrit dans un autre langage. Enfin, l'usage des pointeurs et structures permettent la manipulation efficace de structures de données évoluées. Une discussion détaillée concernant les caractéristiques et les avantages du langage C se trouve dans Kernigham et Ritchie (1988).

Notons aussi qu'il existe de très bons compilateurs C peu coûteux (environ une centaine de dollars) disponibles pour les micro-ordinateurs utilisant le système d'exploitation DOS.

Pour ces raisons, nous avons opté de mettre en oeuvre l'outil TACTIC en langage C. Malheureusement, ceci ne s'effectue pas sans heurt car nous devons reprendre les programmes (FORTRAN) du noyau méthodologique en langage C. Cet effort supplémentaire est largement compensé par la facilité que nous avons à implanter les structures de données complexes du modèle

(e.g. services, itinéraires) avec ce nouveau langage. (Nous discutons de l'implantation des programmes graphiques à la prochaine section.) De plus, les résultats d'expérimentations numériques montrés au chapitre 4 indiquent une amélioration non négligeable au niveau des temps d'exécution.

En conclusion, le choix de ce langage a été judicieux, car il a permis d'écrire des fonctions efficaces et transportables. Soulignons qu'en plus de fonctionner sur un ordinateur Sun utilisant un système d'exploitation UNIX, les modules graphiques de TACTIC ont aussi été implantés à l'École Centrale de Paris (dans le cadre d'un stage parrainé par un projet intégré de coopération France-Québec), en Turbo-C sur un micro-ordinateur IBM-AT et sur un micro-ordinateur IBM-XT muni d'une carte coprocesseur DSI-32 de la compagnie DEFINICON. Ces implantations ont nécessité une modification mineure dans un fichier en-tête pour tenir compte du système d'exploitation DOS.

7.5 Système graphique de base

Les images générées par l'outil TACTIC sont principalement des représentations graphiques de réseaux. Les éléments de base d'un tel dessin sont les noeuds et les liens du réseau, possiblement montrés avec des attributs graphiques ou numériques. Le dessin type est assez simple pour ce qui est de la complexité de la représentation graphique. Par exemple, nous ne manipulons pas d'objets tri-dimensionnels complexes. Par contre, pour ce type de dessin, il faut distinguer les deux échelles suivantes : l'échelle cartésienne (emplacement des objets dans le plan) et l'échelle des attributs (taille des symboles dessinés). Deux discussions plus approfondies sur ce sujet se trouvent dans spiess (1984) et Horowitz (1987).

Pour faire suite au choix du langage de programmation et au choix des équipements graphiques, nous avons mis au point une nouvelle bibliothèque de fonctions graphiques écrites en langage C. Elle porte le nom BGC (Bibliothèque Graphique en C) et elle est bien adaptée au graphisme de réseau.

Un autre avantage lié à l'écriture de notre propre système graphique est le contrôle que nous gardons sur les routines. Cela nous assure la portabilité et le choix des développements futurs de l'outil TACTIC.

Dans ce qui suit, nous ne faisons qu'énoncer les concepts de base sous-jacents à la mise en oeuvre de la bibliothèque graphique BGC. Nous donnons un aperçu des fonctions les plus importantes et nous insistons sur les améliorations que nous apportons, dans ce mémoire, à la première version de BGC (voir Mondou et al., 1987). Enfin, nous référons le lecteur à Mondou et al. (1987) pour la description exhaustive des primitives graphiques de BGC.

7.5.1. Concepts de base

Une bibliothèque graphique est employée par un programme d'application de la même manière qu'un programme utilise les fonctions d'entrée-sortie du système d'exploitation. Ces fonctions d'entrée-sortie gèrent la communication entre le programme et l'architecture de bas niveau de l'ordinateur sans que le programmeur s'en soucie. De la même manière, les fonctions d'une bibliothèque graphique permettent l'utilisation des périphériques graphiques sans que le programmeur ait besoin de connaître leur architecture de bas niveau. La bibliothèque graphique est donc l'interface entre un programme d'application de haut niveau et l'équipement graphique.

Le schéma conceptuel d'un système interactif graphique est présenté à la figure 7.4. Il est composé d'un ensemble d'unités physiques ("hardware") et d'un logiciel d'application ("software").

La partie physique du système consiste en un ordinateur principal ("host computer") auquel sont reliées des unités logiques d'entrée-sortie. Il y a cinq grandes classes d'unités logiques d'entrée :

KEYBOARD : clavier alphanumérique pour l'entrée des chaînes de caractères;

BUTTON : touches ou boutons programmables;

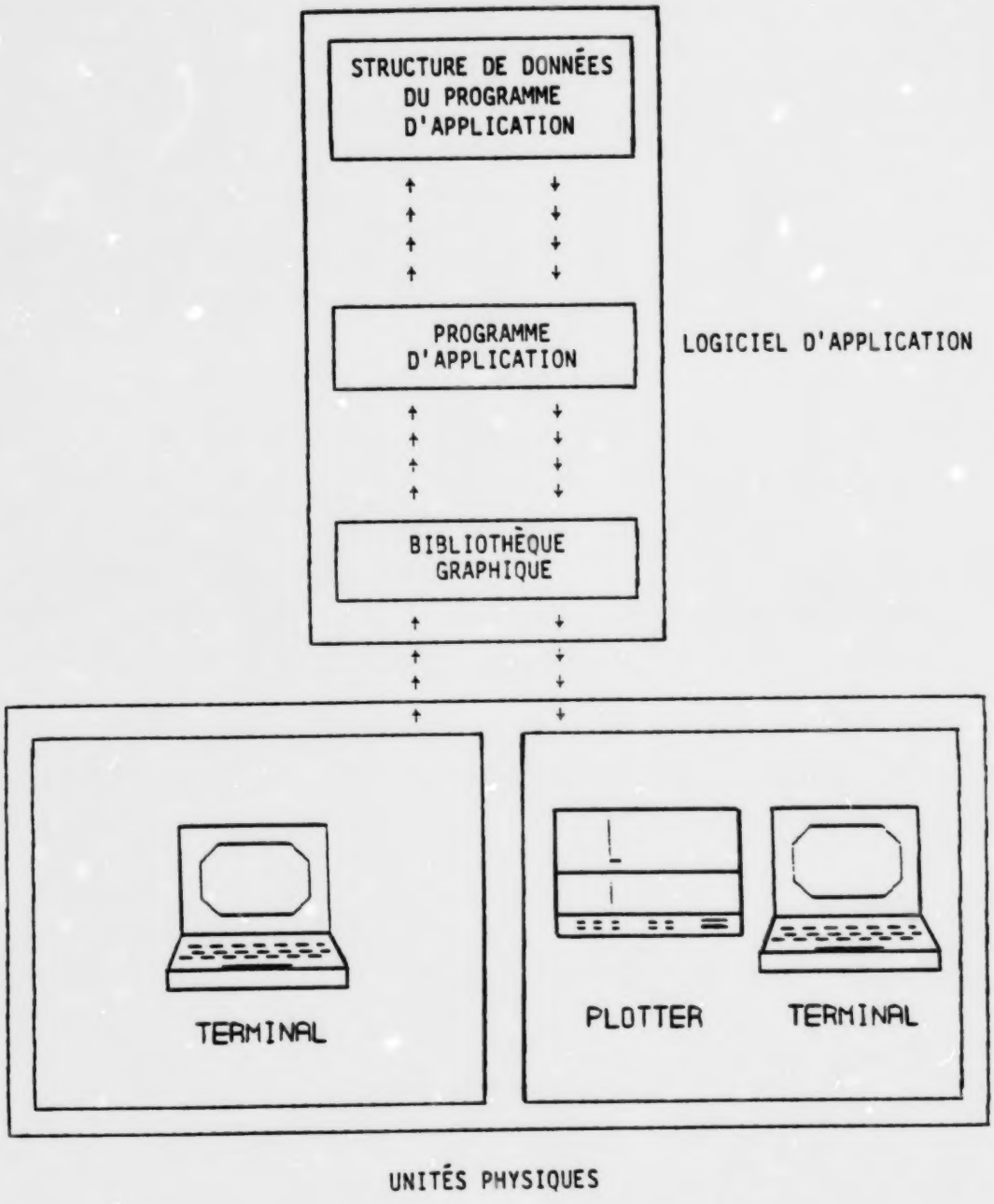


Figure 7.4 : Schéma conceptuel d'un système graphique

PICK : dispositifs pour pointer une information à l'écran (crayon lumineux);

LOCATORS : instruments qui retournent une coordonnée XY de l'écran (souris, joystick, tablette graphique, roulettes, écran tactile, etc.);

EVALUATORS : ce sont généralement des potentiomètres. Ils permettent de spécifier une valeur numérique.

Soulignons qu'il existe aussi d'autres méthodes d'entrée plus sophistiquées telle la reconnaissance de la voix. De plus, des algorithmes ont été élaborées pour simuler une classe logique à l'aide d'un dispositif d'une autre classe. Mentionnons que dans le cadre de notre projet de recherche, nous nous limitons à l'utilisation d'un clavier alphanumérique pour l'entrée des commandes et un mécanisme qui retourne une coordonnée XY de l'écran. Cela suffit à nos besoins actuels.

Pour leur part, les unités de sorties génèrent des images sur une surface de visualisation (écran ou papier). Pour ces unités, nous distinguons deux classes logiques :

TERMINAL : moniteur graphique;

PLOTTER : dispositifs pour l'obtention des copies permanentes (hard-copy). On retrouve dans cette classe les traceurs de courbes, imprimantes, fichier des dessins, etc.

Dans le but de faciliter l'utilisation des équipements graphiques de la classe PLOTTER, nous utilisons un fichier des dessins. Pour BGC, un fichier des dessins est un fichier séquentiel qui accumule des directives graphiques, indépendantes des équipements, reproduisant l'information apparaissant à l'écran. La figure 7.5 résume la syntaxe des commandes graphiques utilisées dans un fichier des dessins de BGC.

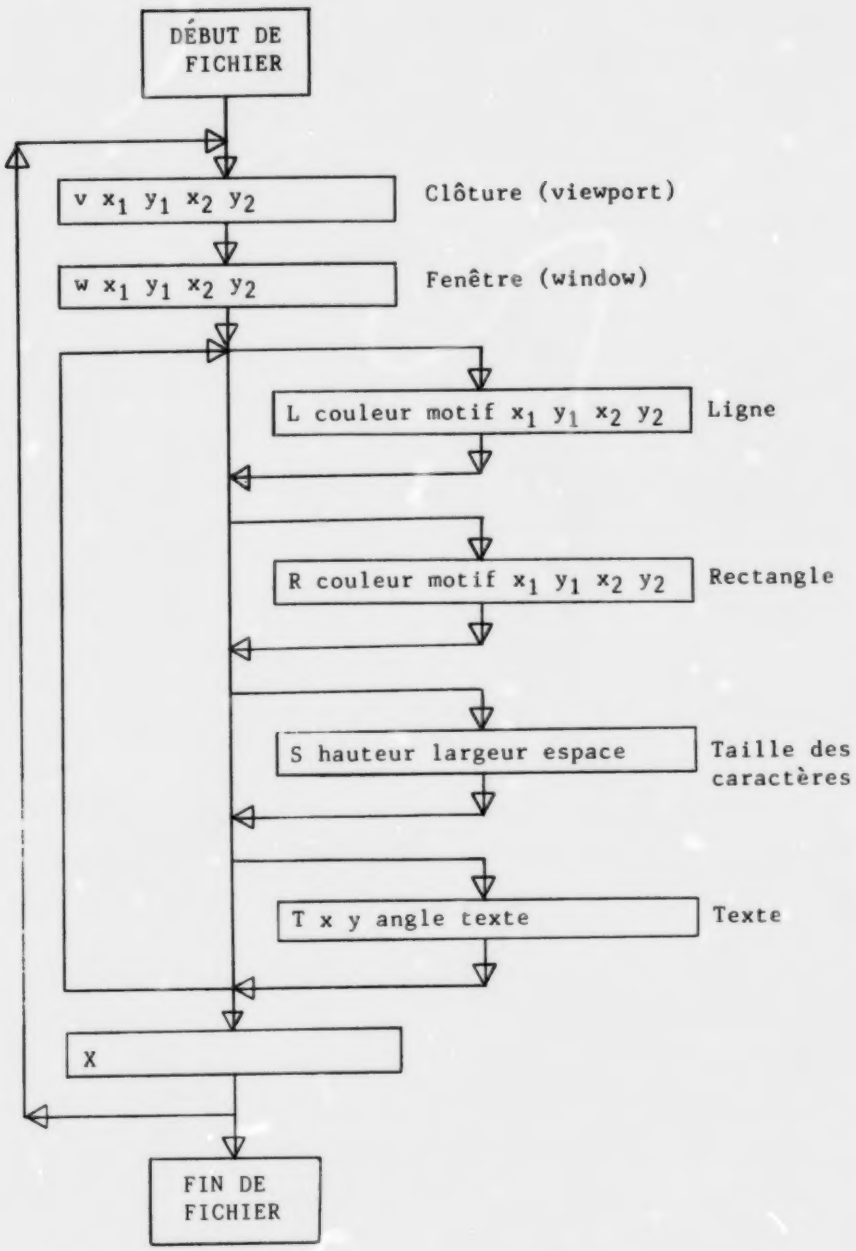


Figure 7.5 : Syntaxe des directives du fichier des dessins

Mentionnons que ce mécanisme de fichier des dessins est différent de celui employé par EMME/2 (Spiess, 1984), STAN (Florian et al., 1987) ou PLANAIR (Lebeuf, 1985). Pour ces logiciels, un fichier des dessins est un fichier qui reçoit les directives propres aux spécifications d'un terminal graphique. C'est aussi l'approche que nous avions adoptée pour la première version de BGC (voir Mondou et al., 1987).

L'utilisation des directives graphiques indépendantes des unités physiques ("graphic device independant") présente l'avantage de pouvoir utiliser tous les équipements, même les cartes vidéo EGA/VGA. Rappelons qu'avec ces cartes, contrairement aux équipements Tektronix, un vecteur est généré par des accès directs à la mémoire vidéo.

De plus, le contenu d'un fichier des dessins peut être envoyé à différentes unités de sorties. Pour ce faire, le fichier est lu par un programme filtre qui génère les commandes compatibles avec un équipement particulier. Les programmes suivants existent déjà :

- dvi24014 : lire "device independant to TEKTRONIX 4014"
- dvi24115 : lire "device independant to TEKTRONIX 4115"
- dvi2ega : lire "device independant to EGA card"
- dvi2hpgl : lire "device independant to HPGL"

Le dernier programme permet d'utiliser les traceurs de la compagnie Helwett-Packard et le traceur Roland GRX-400 disponible au Centre de recherche sur les transports.

Nous tenons à remercier Messieurs Gilles Guénette et Pierre Cossette du Centre de recherche sur les transports d'avoir porté à notre attention l'existence de ces programmes et pour les avoir mis à notre disposition.

Aussi, nous prévoyons écrire le programme - dvi2ps - (lire "device independant to POSTSCRIPT") permettant d'utiliser des imprimantes au laser.

Notons que les copies d'écran incluses dans ce mémoire proviennent d'une imprimante couleur à jet d'encre Tektronix 4696, d'une imprimante à points Gemini 10X et de l'unité "hardcopy" Tektronix 4631.

Finalement, le logiciel d'application comprend trois parties :

- structure de données du programme d'application;
- programme d'application;
- bibliothèque graphique.

La structure de données contient la description des objets réels ou abstraits dont on veut une représentation graphique. Typiquement, un objet est défini par ses coordonnées dans un plan et des attributs décrivant sa taille et sa forme. Nous indiquons aussi le motif et la couleur des vecteurs avec lesquels l'objet est dessiné. Enfin, des liens de connection entre les différents objets sont données pour spécifier leurs relations.

Le programme d'application manipule les objets de la structure de données, gère les interactions usager-ordinateur et envoie les commandes à la bibliothèque graphique. Par exemple, dans ce mémoire, le programme d'application est constitué des modules graphiques de TACTIC.

7.5.2 Structure logique de BGC

La bibliothèque graphique contient un ensemble de fonctions mises à la disponibilité du programme d'application. Nous nous sommes efforcés d'employer un style de programmation clair et modulaire pour faciliter les améliorations et les extensions futures. De plus, nous avons organisé BGC comme une boîte noire, c'est-à-dire qu'elle est indépendante des programmes d'application qui l'utilisent.

La structure de BGC est illustrée à la figure 7.6 et elle est composée de cinq grandes classes de fonctions :

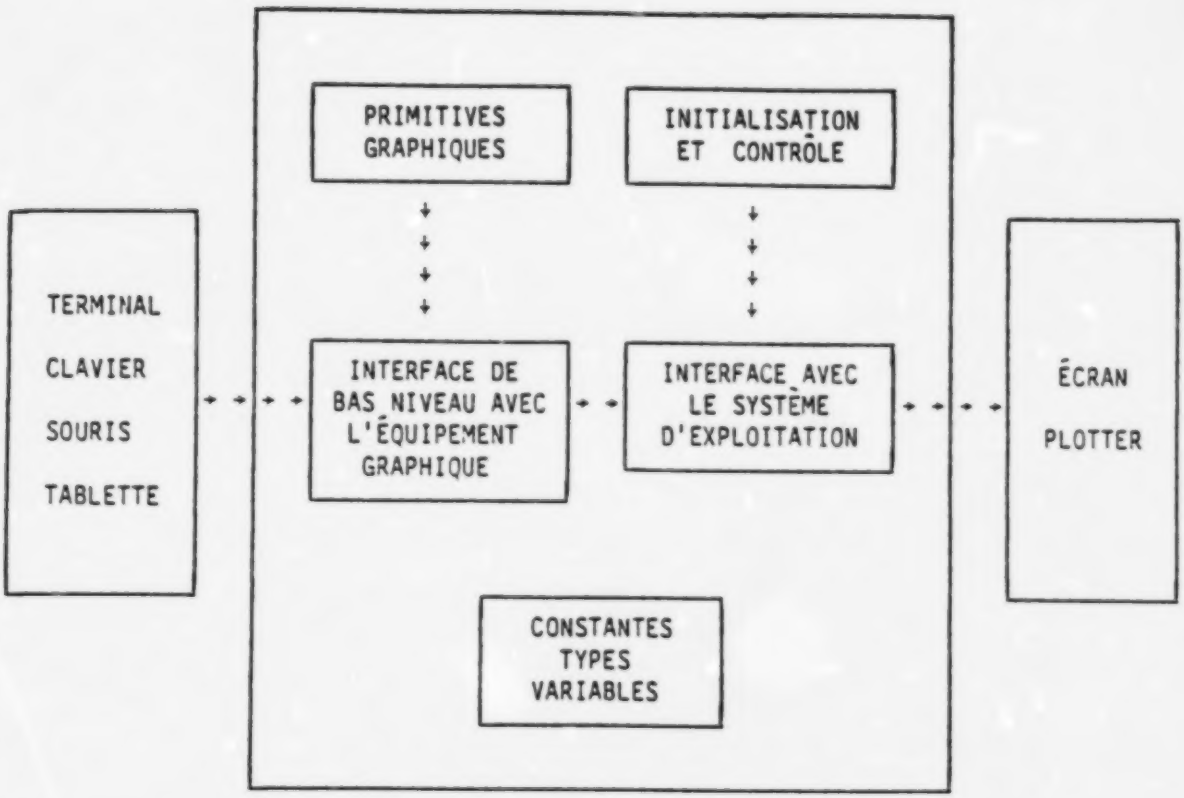


Figure 7.6 : Structure de BGC

1) INITIALISATION ET CONTRÔLE :

Ce sont les fonctions qui initialisent l'équipement pour un nouveau dessin. Pour un terminal graphique, cela implique l'effaçage de l'écran et l'initialisation des paramètres pertinents. D'autres fonctions permettent au programmeur d'utiliser les fonctions du système d'exploitation pour faire sonner la cloche, obtenir la date, effacer l'écran, etc. Il y a aussi une fonction qui imprime la version de BGC et donne la liste des dispositifs graphiques connus.

Pour faire suite aux exigences posées au chapitre 1 concernant le développement de l'outil tactique, la bibliothèque graphique doit permettre l'utilisation de dispositifs graphiques de différents manufacturiers.

Une première approche (Spiess, 1984; Mondou et al., 1987), pour faire face à ce problème, consiste à utiliser des fichiers sur disques qui contiennent des chaînes de caractères à envoyer aux différents terminaux pour les initialisations, et des valeurs numériques spécifiant les caractéristiques de ces terminaux. Malheureusement, il existe des émulateurs imparfaits des standards graphiques et les fonctions doivent contenir des exceptions ("patch") pour les traiter. Par conséquent, les fonctions graphiques de bas niveau (voir section suivante) doivent constamment tester les caractéristiques du terminal afin de générer les bonnes directives graphiques. De plus, nous remarquons que cette façon de faire ne fonctionne pas si les équipements graphiques sont des appareils incompatibles avec le standard Tektronix (e.g. les cartes VGA). Inévitablement, nous devons ajouter des exceptions aux fonctions de BGC et les recompiler lorsque nous désirons employer un nouveau dispositif graphique.

Dans ce mémoire, nous apportons une nouvelle solution au problème. Nous utilisons la variable entière, TERMTYPE, qui prend les valeurs suivantes :

0 si fichier des dessins

1 si Tektronix 4014

3 si Visual 550

5 si Tektronix 4115

Aussi, pour chacun de ces terminaux, nous avons programmé les fonctions de bas niveau (voir section suivante). Au moment de l'initialisation de la bibliothèque graphique, nous plaçons des pointeurs sur les fonctions graphiques de bas niveau selon la valeur de la variable TERMTYPE. Le principal avantage de cette implantation est l'élimination des fréquents tests nécessaires pour traiter les exceptions. En plus, cela ne nous restreint pas à l'utilisation d'un seul standard graphique. Finalement, l'ajout d'un nouveau terminal nécessite la mise en oeuvre de nouvelles fonctions et, comme précédemment, la recompilation des fonctions de BGC.

2) INTERFACE DE BAS NIVEAU :

Ce sont les fonctions qui effectuent les opérations d'interface ("driver") entre les primitives graphiques et les équipements. Principalement, elles envoient les commandes aux unités graphiques pour générer des vecteurs, modifier leurs attributs (e.g. couleur, motif de tracé, etc.) et activer les dispositifs d'entrée graphique. De plus, elles permettent d'utiliser les caractéristiques de ces unités, tel l'effaçage sélectif et les caractères générés par celles-ci. Mentionnons que nous avons doté BGC d'un générateur de caractères pour les équipements qui ne peuvent pas générer des caractères dans n'importe qu'elle direction et taille (e.g. Visual 550, Tektronix 4014). Ce générateur de caractères emploie les tables de coordonnées Hershey (Wolcott et Hilsenrath, 1976).

Pour ces fonctions d'interface, nous avons fixé arbitrairement un écran graphique normalisé dont les dimensions sont les suivantes :

$$0.0 < x < 4096.0 ,$$

$$0.0 < y < 3120.0 .$$

Le ratio d'aspect est d'environ 4X:3Y, le format trouvé dans la plupart des terminaux graphiques contemporains. Ce système de coordonnées logique recouvre toute la surface de visualisation des unités graphiques employées et possède l'avantage d'être indépendant de ceux-ci. Cet écran définit la clôture ("viewport") maximale qui est utilisée pour les opérations de visualisation. Mentionnons que le choix de ces dimensions ne limite pas la précision du dessin, car elles admettent des valeurs réelles. C'est à la dernière étape qu'elles sont transformées et tronquées pour tenir compte de la résolution de l'équipement. Nous soulignons que nous avons fixé les dimensions de l'écran normalisé afin de minimiser l'effort de calcul pour tenir compte de la résolution des terminaux graphiques Tektronix 4115 et Tektronix 4014.

Enfin, les fonctions de cette classe sont internes à BGC et ne peuvent pas être appelées par le programme d'application. Cela crée l'effet "boîte noire" de BGC.

3) MODIFICATION DES ATTRIBUTS :

Ces fonctions gèrent la valeur courante du motif de tracé des vecteurs (variable entière LINESTYLE) et leur couleur (variable entière COLOR). Pour affecter une valeur à ces variables, le programme d'application fait appel aux fonctions set-lstyle () et set-color (). De plus, les fonctions get-lstyle () et get-color () retournent respectivement la valeur des variables LINESTYLE et COLOR. Ce mécanisme est nécessaire afin de conserver l'effet "boîte noire" de BGC. Nous avons aussi les variables suivantes :

CHEIGHT : "character height" ou hauteur des caractères[1],
CWIDTH : "character width" ou largeur des caractères[1],
CSPACING : "character spacing" ou espace entre les caractères[1],

---

[1] En unité de l'écran normalisé.

FPATTERN : "fill pattern" ou motif de remplissage,

HATCH : espacement entre les vecteurs de remplissage[1],

SCALE : échelle, nombre d'unités de l'attribut par unité de coordonnée normalisée,

BCKG-R : niveau de rouge pour la couleur de fond,

BCKG-G : niveau de vert pour la couleur de fond,

BCKG-B : niveau de bleu pour la couleur de fond,

et leurs fonctions associées.

4) OPÉRATION DE VISUALISATION :

Les objets à afficher sont définis par des coordonnées arbitrairement choisies par le programmeur. En général, il s'agit du même système de coordonnées qui a été utilisé pour géocoder le réseau. BGC doit projeter ces coordonnées dans l'univers de l'écran normalisé pour obtenir un dessin à l'unité graphique. Pour effectuer cette conversion, BGC doit connaître la portion du système de coordonnées du programmeur qui contient l'information à être dessinée à ce moment. Cette région rectangulaire de l'univers de l'usager est une fenêtre ("window").

Il ne faut pas confondre cette terminologie dans le contexte graphique avec la définition présentée dans la documentation des interfaces orientées-écran. Pour ces systèmes, la fenêtre désigne une région de l'écran. Pour la bibliothèque graphique, une clôture ("viewport") est une région rectangulaire de l'écran normalisé dans laquelle les informations de la fenêtre sont affichées.

Les fonctions de cette classe effectuent les manipulations sur les fenêtres et les clôtures. Aussi, elles accomplissent les opérations de découpage sur les vecteurs ("clipping").

---

[1] En unité de l'écran normalisé.

5) PRIMITIVES GRAPHIQUES :

C'est le niveau hiérarchique le plus élevé de BGC. Ce sont des fonctions indépendantes du système d'exploitation et des équipements graphiques.

Elles permettent de dessiner des vecteurs, des rectangles, des cercles, des flèches, des chaînes de caractères, des noeuds et des liens du réseau, etc. Nous rappelons que le motif et la couleur de tracé des vecteurs qui composent ces primitives graphiques sont ceux définis par les fonctions set-lstyle () et set-color (). La figure 7.7 illustre quelques exemples de dessins produits par les primitives graphiques de BGC.

7.6 Utilisation de la couleur

L'utilisation de la couleur dans l'affichage graphique présente un nombre d'avantages incontestables dont, outre des présentations visuellement plus intéressantes et dynamiques, plus de souplesse dans la représentation et une meilleure discrimination entre objets et éléments d'information.

D'un autre côté, l'équipement nécessaire est généralement, soit plus dispendieux, soit de moins bonne résolution que les terminaux monochromes correspondants. De plus, la "couleur" est une caractéristique très personnelle pour chaque type de terminal, d'où la nécessité de définir des tables de couleurs et de possibles problèmes de portabilité des logiciels si les couleurs reçoivent des significations propres permanentes.

Nous avons donc décidé d'adopter une approche mixte. Nous utilisons la couleur seulement pour distinguer les objets, sans lui accorder de signification propre absolue. En l'absence de couleur, nous utilisons des formes et symboles géométriques.

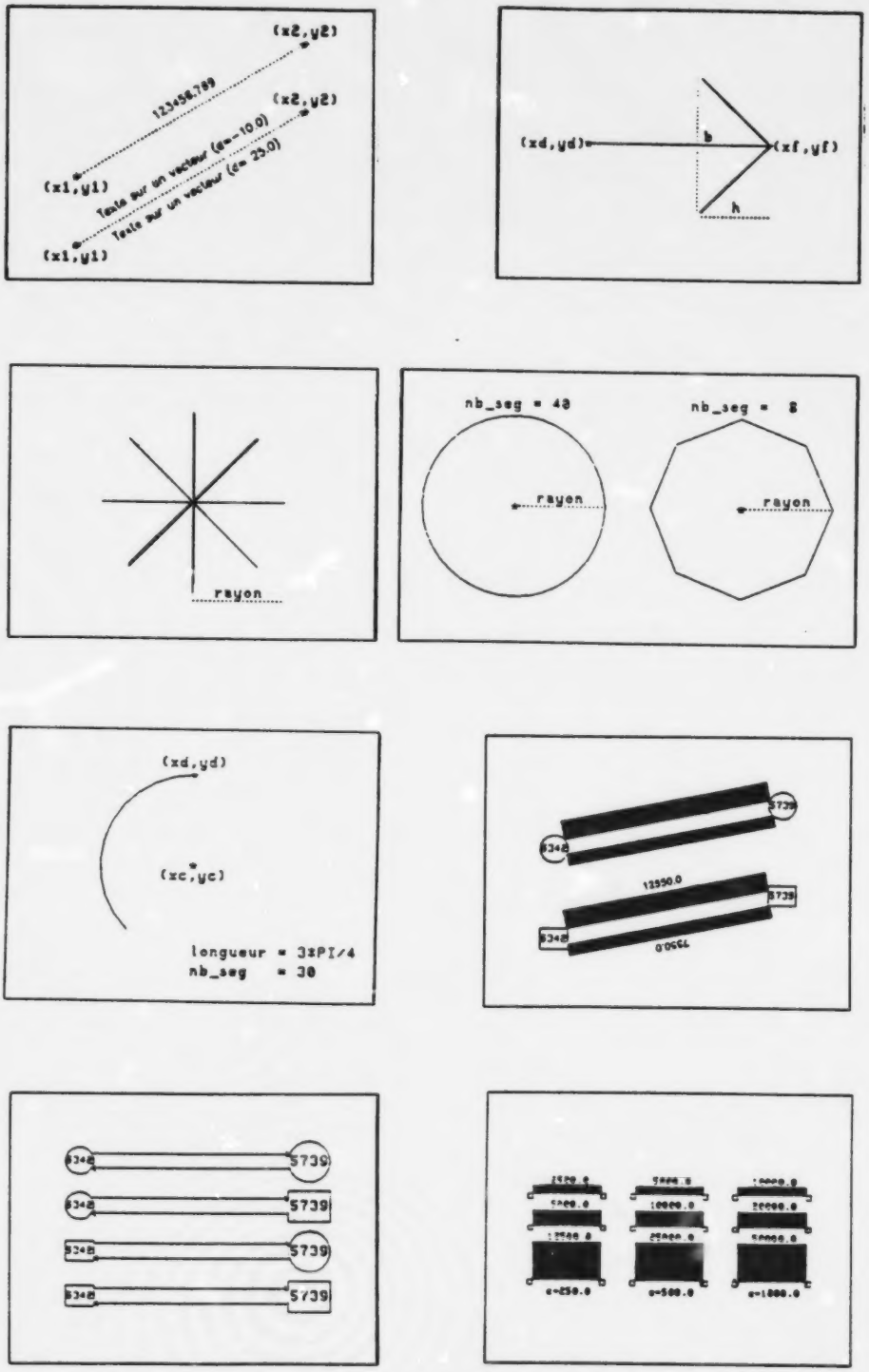


Figure 7.7 : Quelques dessins obtenus à l'aide des primitives graphiques BGC

7.7 Représentation et affichage

La complexité du transport des marchandises se reflète dans la diversité et le nombre élevé d'informations qui doivent être affichées, souvent sur un même écran.

Les problèmes à résoudre proviennent principalement de deux sources: la représentation "chemin" de l'acheminement du fret et des services de transport et la nécessité de représenter sur un même élément (exemple : un lien physique) plusieurs caractéristiques à composantes multiples (exemple: nombre de convois par type de service, tonnage total par classe de marchandise et nombre de véhicules par classe de marchandise, etc.).

7.7.1 Les éléments à afficher

Voici une énumération des divers éléments du système, avec leurs attributs, exigences et possibilités d'affichage.

A) Réseau physique

- Description : noeuds et liens, avec identification, types, caractéristiques physiques;
- Utilisation : flots totaux de marchandises, véhicules, convois, etc.

B) Analyse par élément

B.1) Liens

- Nombre de convois par type de service;
- Nombre de
  - . tonnes
  - . véhicules
  - . blocs

  par
  - . classe de marchandise
  - . type de service
  - . marché.

B.2) Noeuds

- Nombre de convois par mode
  - . créés
  - . terminés
  - . passant;

- Nombre de blocs créés par destination;
- Nombre de
  - . véhicules
  - . tonnes

  qui
  - . originent
  - . terminent
  - . sont classés
  - . sont transférés

  par
  - . destination ou origine
  - . classe de marchandises
  - . type de trafic.

C) Réseau de service

- Représentation : physique (la route physique est affichée) ou conceptuelle (seulement les arcs de service sont représentés);
- Affichage par
  - . service
  - . paire origine-destination
  - . réseau entier;
- Description
  - . route physique
  - . segments de service
  - . arrêts intermédiaires
  - . mode-type de service
  - . fréquence;
- Utilisation d'un service en
  - . tonnage
  - . nombre de véhicules
  - . nombre de blocs

  par
  - . marché
  - . paire origine-destination
  - . classe de marchandises;
- Attributs du service : coûts et temps de parcours par route, segment de service ou lien.

D) Trafic des marchandises

D.1) Représentation

- Demande : matrices origine-destination et aggrégations;
- Mouvements : les itinéraires;
- Mesuré en
  . tonnes
  . véhicules;
- Affiché par
  . marché
  . paire origine-destination
  . classe de marchandises
  . type de trafic;

D.2) Affichage des itinéraires

- sur
  . réseau physique
  . réseau de service;
- identifiant
  . terminus de travail
  . blocs
  . services;
- inscrivant
  . coûts
  . délais
  . flots.

Cette énumération, bien que non exhaustive, couvre les principales utilisations pratiques. D'ailleurs, toutes les combinaisons ne sont pas nécessaires pour chaque application. Nous avons cependant cherché des solutions générales qui puissent s'adapter facilement aux exigences particulières de chaque application.

Les trois prochaines sous-sections sont consacrées à l'étude des problèmes reliés à l'affichage des divers éléments du système que nous venons d'énumérer ainsi que les solutions que nous y apportons. Notre discussion est illustrée à l'aide de plusieurs sorties graphiques dont les données proviennent d'une application réelle.

Le problème qui sert à notre expérimentation examine la planification du transport des marchandises sur la partie ouest du réseau de la compagnie des Chemins de fer Nationaux (C.N.). Soulignons que le problème de la planification tactique des systèmes de transport ferroviaire des marchandises figure parmi les applications les plus complexes de ce type de planification (voir Crainic, 1982). Voici les caractéristiques de notre problème test.

En appliquant des règles d'agrégation propres au C.N., nous obtenons un réseau composé de 73 noeuds et 180 liens.

Parmi les noeuds, trois représentent des points de jonction (JUNCTION) physique des voies ferrées, où il n'y a aucune activité. Les autres noeuds représentent soit une gare de triage (TERMINAL) ou une station (STATION). Les stations sont des gares du réseau physique où les trains peuvent prendre ou livrer de la marchandise. À ces gares, il est aussi possible d'effectuer le transfert de wagons d'un service de train à un autre. Pour chacune des gares de triage, nous connaissons la capacité maximum de classification ainsi que le coût moyen de classification par wagon. Les noeuds de ce réseau ont été géocodés au Centre de recherche sur les transports à l'aide du traceur/digitaliseur Tektronix 4663.

Parmi les liens, neuf représentent des voies doubles, toutes situées sur la ligne principale du C.N. Pour chaque lien, nous connaissons sa longueur, le nombre de voies d'évitement, ainsi que le nombre de trains de passagers qui y passent.

La demande de transport spécifiée sur ce réseau représente les prévisions pour le quatrième trimestre de l'année 1991. La période de planification utilisée est la semaine. Il y a 611 marchés et 33 groupes de marchandises. Le réseau de service comporte 191 services et 415 arcs de service. Pour chacun des trois types de services autorisés (rapide, lent et

unitaire), nous connaissons sa vitesse, ainsi que les règles de priorité lors des rencontres et dépassements. Les coûts de délai et d'opération proviennent des tables et formules du C.N.

Ce problème a été résolu avec l'algorithme mis au point par Crainic (1982) et il a généré 1055 itinéraires optimaux.

7.7.2 Affichage du réseau physique

Le réseau physique complet est montré à la figure 7.2 (voir section 7.2). Nous constatons que les détails de la partie centrale de cette figure, représentant les provinces des prairies, se perdent dans un amas de noeuds et il est impossible d'interpréter le dessin. Pour pallier à cette difficulté, nous agrandissons cette partie du dessin ("zoom in"). Cette opération s'appelle prendre une fenêtre. Le résultat est montré à la figure 7.8.

Sur cette figure, les noeuds du réseau sont représentés par des rectangles ou des cercles de différentes grosseurs et différentes couleurs selon le choix de l'utilisateur. Mentionnons que cela peut être modifié à tout moment pendant l'exécution des modules graphiques. Pour cette application de transport des marchandises par chemin de fer, il y a deux types de liens : les voies simples et les voies doubles. Soulignons que cela varie selon les applications. Comme pour les noeuds, l'utilisateur a la possibilité de sélectionner les attributs (motif de tracé, couleur, flèche, espacement) pour dessiner les liens.

Toujours à la figure 7.8, dans le coin inférieur gauche, nous avons dessiné le squelette du réseau physique complet ainsi que les contours géographiques et politiques du Canada. Cette portion du dessin, qui apparaît sur demande, sert à indiquer la fenêtre dont le contenu est dessiné dans la partie principale de l'écran. Elle peut aussi être utilisée pour activer une nouvelle fenêtre, n'importe où sur le réseau.

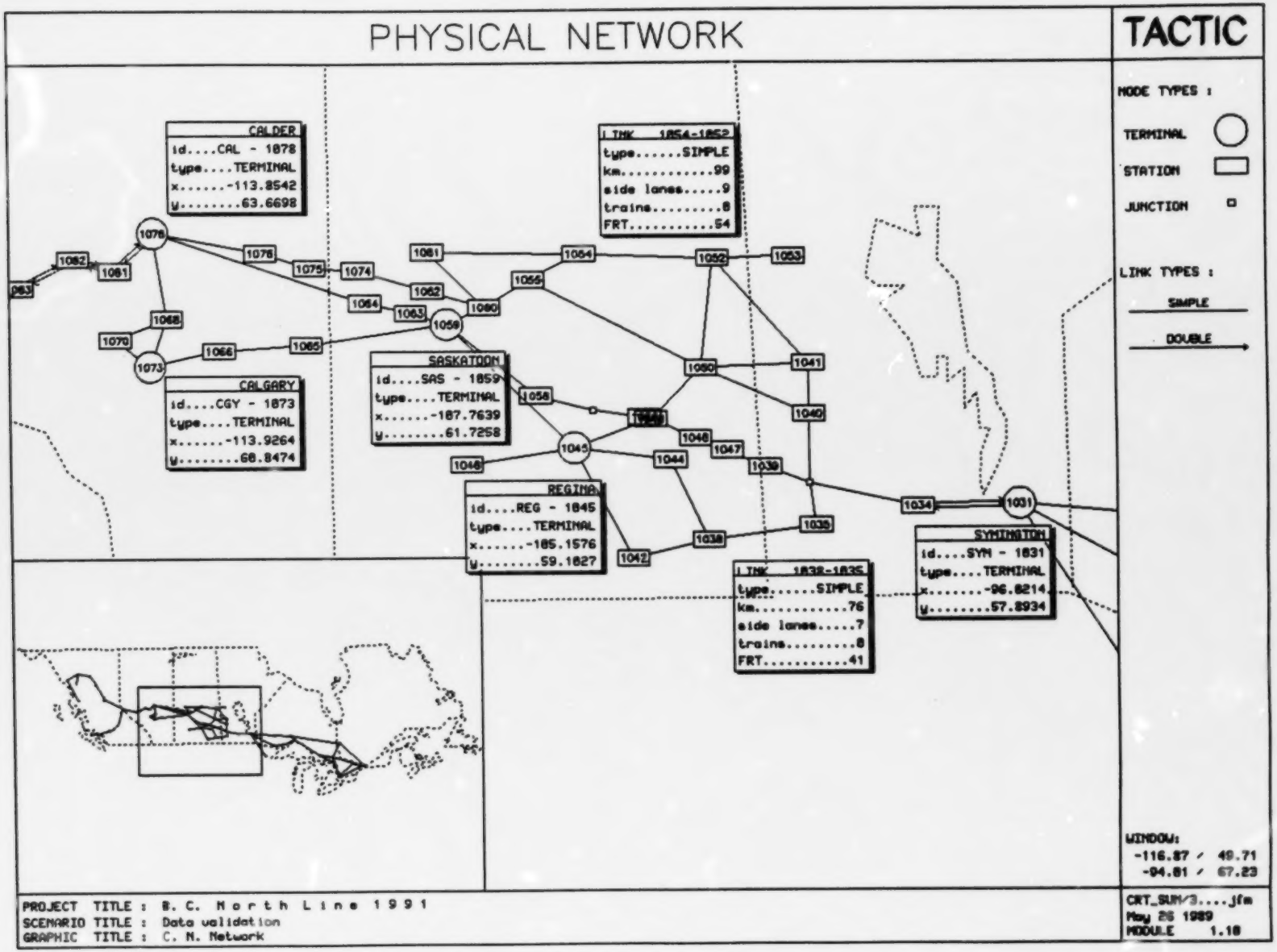


Figure 7.8 : Réseau physique avec quelques caractéristiques physiques

De plus, afin de consulter les caractéristiques physiques associées à un noeud ou à un lien, l'utilisateur peut pointer l'élément en question avec un dispositif d'entrée graphique (voir section 7.5.1), un lien étant spécifié par son noeud origine et par son noeud destination, pour faire apparaître ces informations dans une petite boîte. La position de la boîte est aussi déterminée par l'utilisateur. Notons que les informations contenues dans les boîtes de la figure 7.8 sont données à titre d'exemple. Dans une étape ultérieure du projet, nous ajouterons d'autres renseignements. Cette méthode d'affichage des informations est élégante et pratique et nous l'utilisons aussi dans d'autres contextes (voir plus loin).

Mentionnons que ces deux dernières caractéristiques graphiques sont propres à l'outil TACTIC.

Les figures 7.9 et 7.11a sont des exemples de représentation graphique d'une mesure totale (e.g. flux total de marchandises) d'utilisation du réseau. La figure 7.9 montre le flux total de marchandises circulant sur les liens du réseau physique. Sur cette figure, la valeur du flux, x, est représenté par une barre dont l'épaisseur, BWIDTH, en nombre d'unité de l'écran normalisé, est calculée par la formule simple :

$$\text{BWIDTH} = \frac{X}{\text{SCALE}} .$$

Rappelons que pour les dessins de réseau, nous distinguons l'échelle des attributs (taille des symboles dessinés) et l'échelle cartésienne (emplacement des objets). Pour cette raison, la valeur de la variable BWIDTH est donnée en unités de l'écran normalisé afin de demeurer indépendante des coordonnées de la fenêtre. Une discussion plus approfondie sur ce sujet se trouve dans Spiess (1984) et Horowitz (1987). La valeur du flot peut être représentée de trois façons, au choix de l'usager, illustrées à la figure 7.10. Par convention, nous avons fixé que le flux sur un lien allant de i vers j se trouve toujours à gauche. Enfin, l'utilisateur peut spécifier l'espacement entre les vecteurs de remplissage (HATCH).

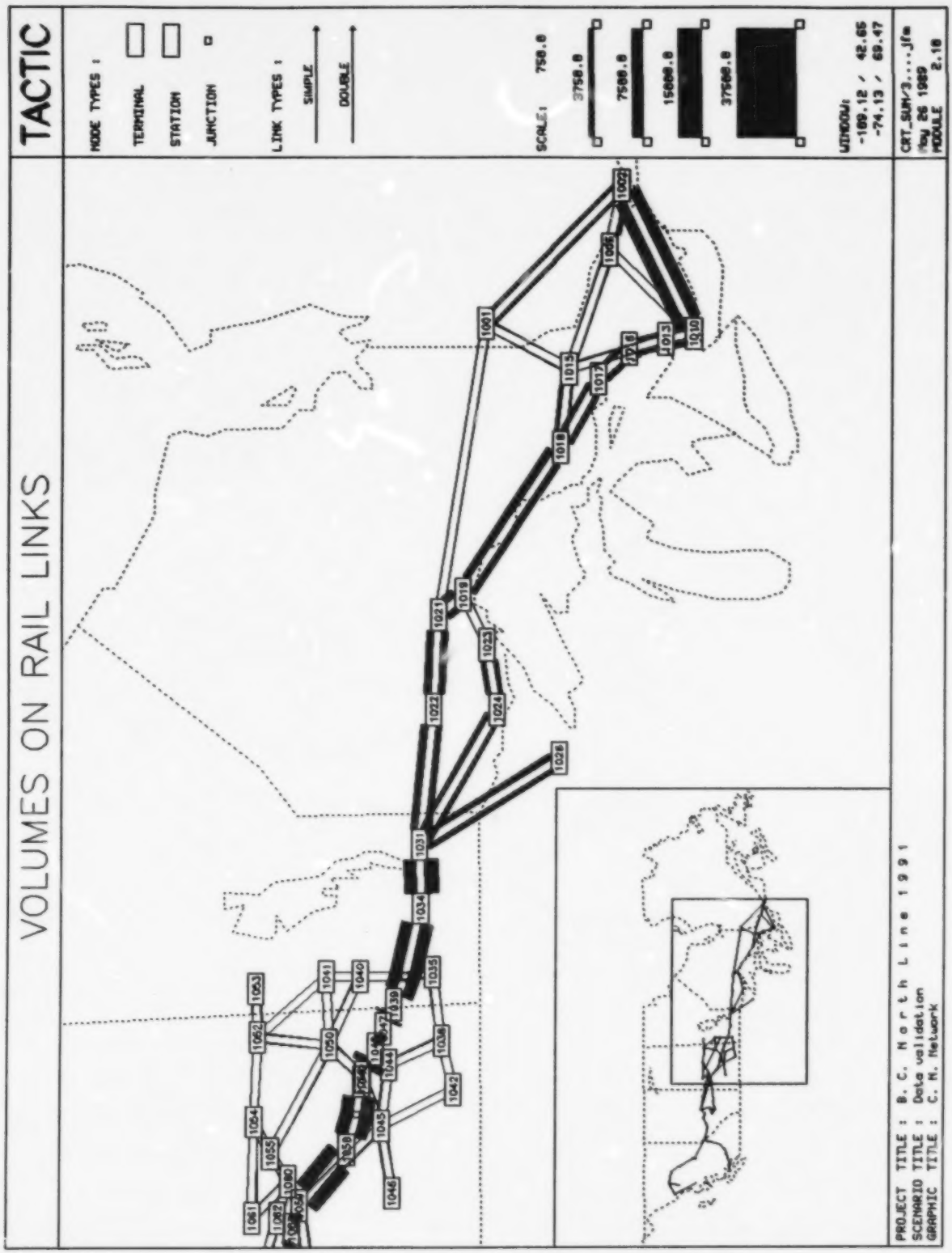


Figure 7.9 : Réseau physique avec flux total de marchandises sur les liens

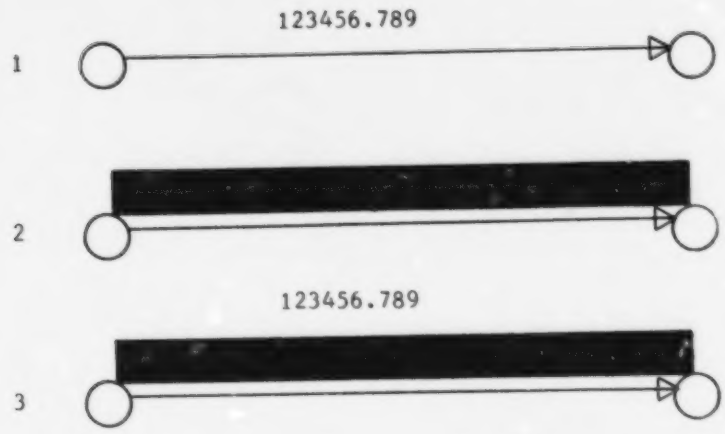


Figure 7.10 : Représentation du flux sur un lien

La figure 7.11a montre le flux total de marchandises traitées aux noeuds du réseau physique. Sur cette figure, la valeur du flux est représentée par un cercle dont le rayon, mesuré en unités de l'écran normalisé, vaut la valeur du flux divisé par la valeur de l'échelle. Comme précédemment, l'utilisateur peut contrôler l'espacement entre les barres de remplissage. Actuellement, c'est la seule façon de visualiser un flux à un noeud. Cependant, nous prévoyons aussi utiliser le symbole (rectangle, cercle) représentant le noeud pour y inscrire la valeur numérique du flux, à la place de l'identificateur.

7.7.3 Analyse par élément

Le problème posé est celui de pouvoir représenter clairement la distribution sur chaque élément du réseau (liens ou noeuds) d'une ou plusieurs mesures d'utilisation du réseau pour un même attribut (tel, pour les liens, le nombre de véhicules par classe de marchandises, le tonnage

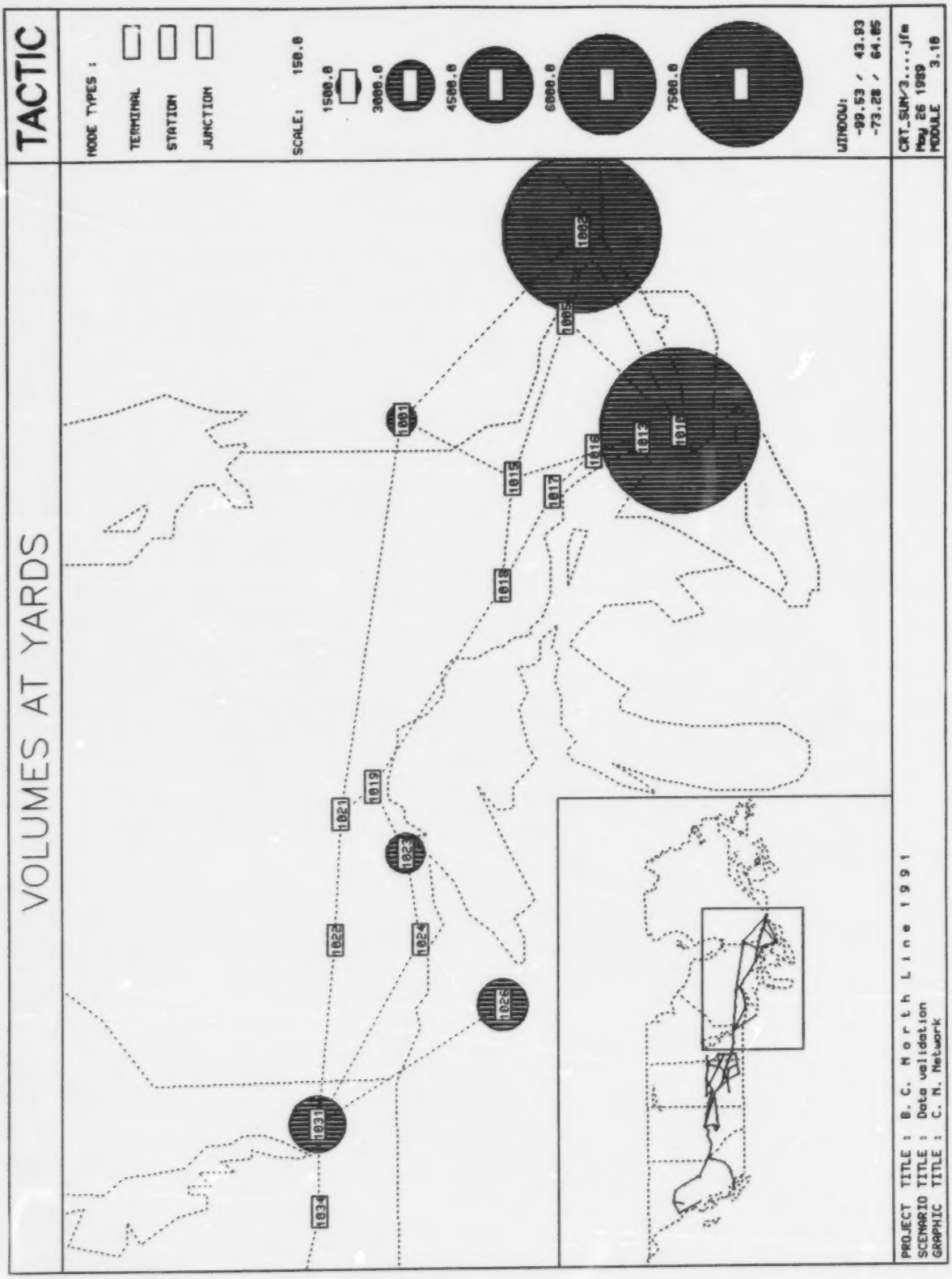


Figure 7.11a : Réseau physique avec flux total de marchandises aux noeuds

par classe de marchandises, voir section 7.7.1). Le problème d'affichage est compliqué par le nombre de mesures possibles, le grand nombre de cas possibles pour certains attributs (il peut y avoir environ 30 classes de marchandises, par exemple) et par la densité du réseau.

En premier lieu, nous avons examiné les possibilités d'affichage de la distribution d'une mesure pour un attribut sur les liens du réseau (e.g. le nombre de véhicules par type de marchandise).

Deux possibilités ont retenu notre attention.

La première, illustrée à la figure 7.11b, affiche l'histogramme de la mesure sur chaque lien, dans les deux sens, si nécessaire. Nous remarquons qu'il y a très vite perte de clarté par superposition d'histogrammes lorsque le nombre de cas possibles augmente, lorsque la densité du réseau augmente ou lorsque les arcs sont presque parallèles.

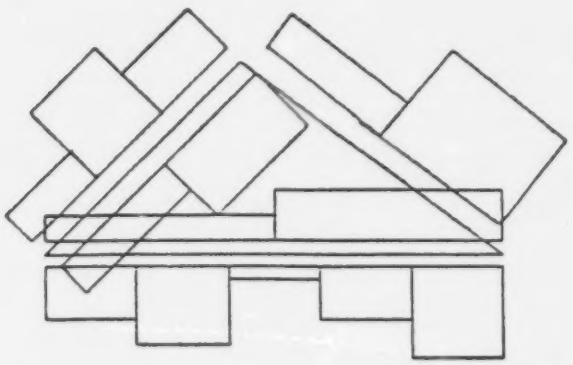


Figure 7.11b : Une mesure pour un attribut, par lien, en histogramme

Une deuxième possibilité, voir figure 7.12, utilise la couleur pour décrire la valeur de la mesure. À chaque lien, nous associons une bande divisée en autant de parties qu'il y a de cas possibles pour la mesure affichée, la couleur de chaque sous-division indiquant l'intensité de

l'utilisation de la mesure. Cette possibilité présente l'avantage d'un moindre encombrement du réseau dû à la largeur uniforme des bandes associées aux liens. Cependant, cette méthode requiert un terminal couleur car l'utilisation de symboles géométriques, des hachures par exemple, pour remplacer la couleur amène une perte de clarté du dessin d'autant plus rapide que le nombre de cas augmente. Pour cette raison, nous avons rejeté cette approche car elle entrave la portabilité du système.

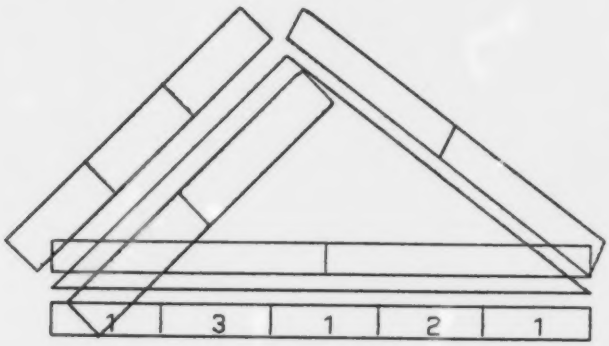


Figure 7.12 : Une mesure pour un attribut, par lien, en couleur

Nous avons aussi examiné plusieurs possibilités d'affichage de deux ou plusieurs mesures à la fois par attribut sur les liens du réseau dont :

- histogramme à trois dimensions (voir figure 7.13) pour le cas des deux mesures;
- association à chaque arc d'autant de bandes à sous-divisions de couleur que de mesures à représenter;
- représentation simultanée à l'écran de plusieurs copies du réseau, chaque copie affichant une mesure.

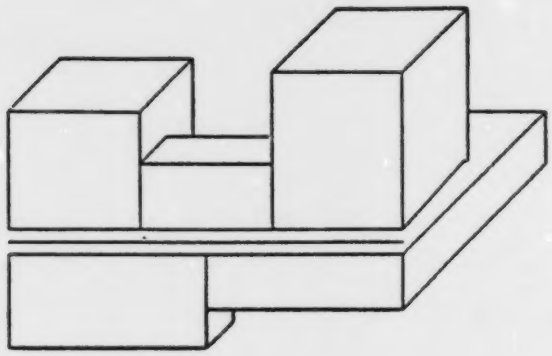


Figure 7.13 : Deux mesures pour un attribut, par lien, en histogramme à trois dimensions

Chacune de ces alternatives possède certains avantages, mais toutes génèrent des problèmes inacceptables d'encombrement de l'image et de généralisation à des dimensions supérieures.

Nous avons donc adopté les solutions suivantes, illustrées aux figures 7.14, 7.15 et 7.16 : représenter un élément et une mesure pour un attribut à la fois, en utilisant des histogrammes (et la couleur quand elle est disponible), mais représenter plusieurs combinaisons élément-mesure par écran ou par page. Ainsi, l'utilisateur choisit tant les éléments que les mesures à afficher et, pour fins d'impression, le nombre de combinaisons à mettre par page. Sur la figure 7.14, par exemple, le nombre de combinaisons affichées sur un écran a été fixé à quatre pour des raisons de clarté visuelle. Pour des applications et de l'équipement différent, ce nombre est facilement modifiable. Un nombre variable (à être défini dans certaines limites par l'usager) de telles zones est aussi prévu. Aussi, remarquons que l'outil permet de définir des fenêtres dans toutes les zones, permettant ainsi d'obtenir le degré de détail souhaité. L'autre implantation consiste à travailler directement sur le réseau. Dans ce cas, l'utilisateur pointe un élément, choisit les mesures à afficher et il indique l'endroit

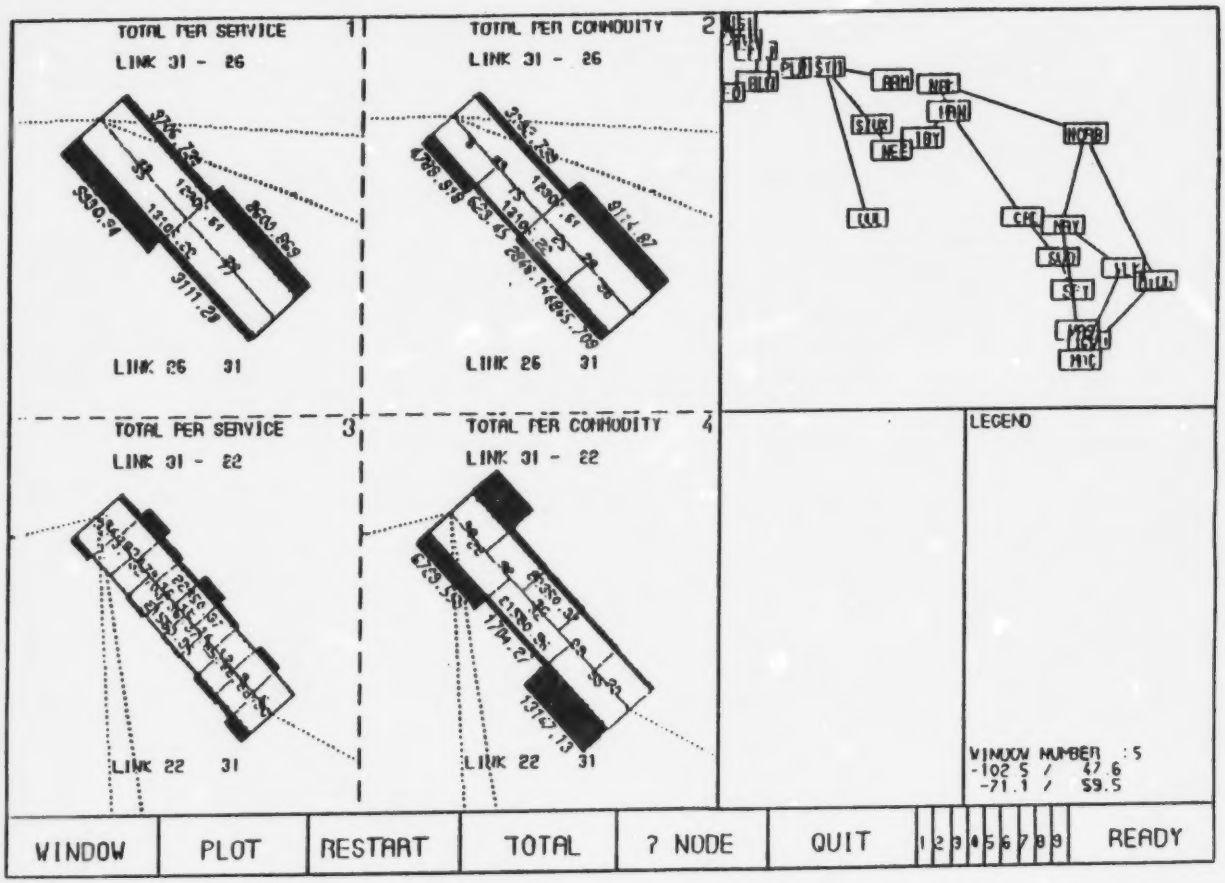


Figure 7.14 : Analyse par lien - deux mesures

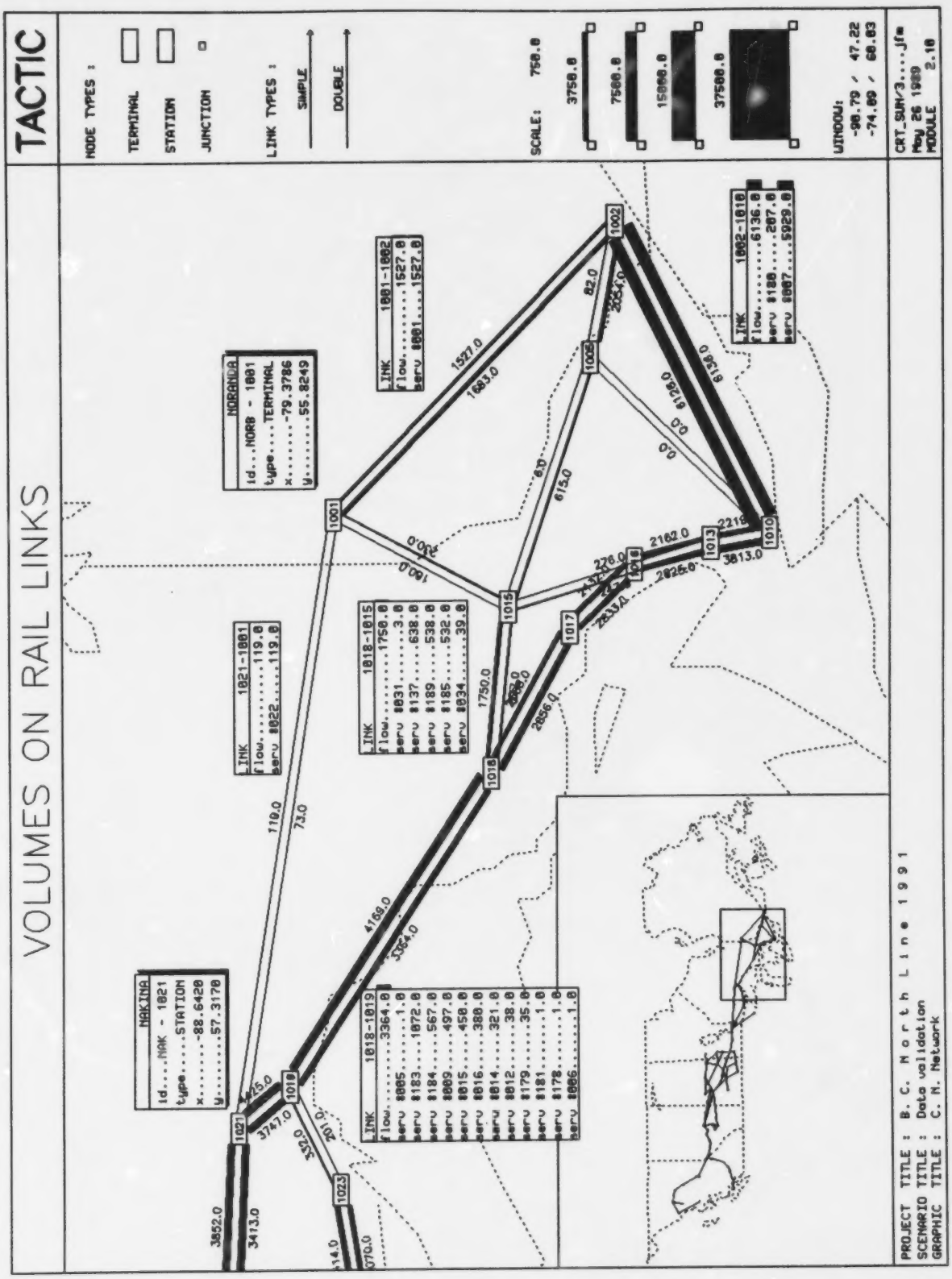


Figure 7.15 : Analyse par lien - deux mesures - autre représentation

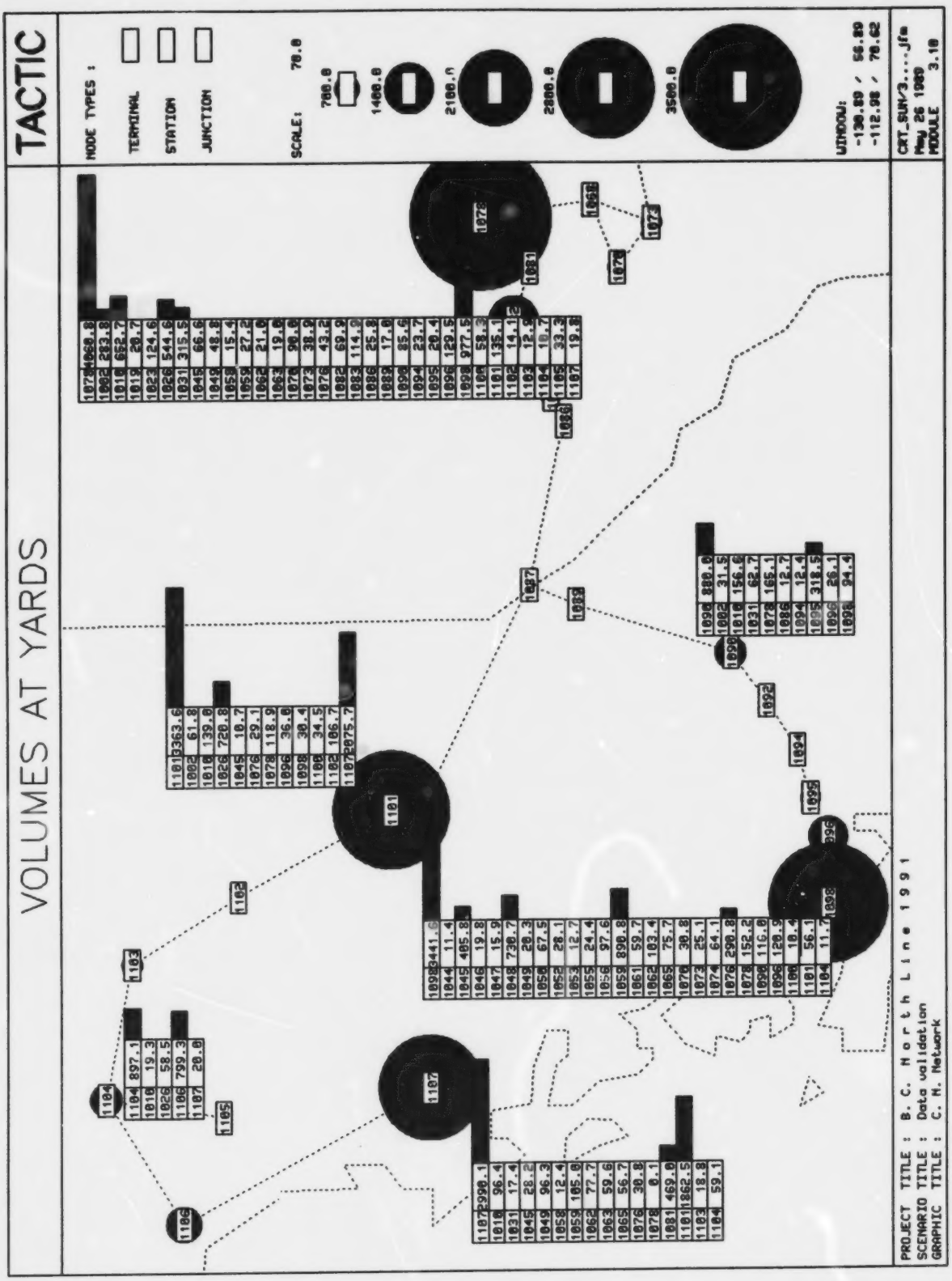


Figure 7.16 : Analyse par noeud - deux mesures

sur le dessin où elles doivent apparaître. Cela est montré aux figures 7.15 et 7.16. Cette façon de faire ressemble à ce que nous avons présenté à la section précédente pour l'affichage des caractéristiques physiques du réseau (voir figure 7.8).

Cette approche est générale, dans le sens où elle permet de réaliser l'analyse de l'utilisation tant des noeuds-terminus que des liens, et ce, par rapport à un nombre quelconque de mesures. Elle est aussi facilement généralisable aux cas de l'analyse individuelle d'un service, d'un itinéraire ou d'un marché.

7.7.4 Représentation "chemin"

Nous rappelons qu'un service est caractérisé, dans le réseau physique, par une gare origine et une gare destination, une route les reliant et, éventuellement, des arrêts intermédiaires dans certaines gares sur la route. Le problème de la représentation des services est comparable à celui de l'affichage des lignes de transport en commun (Spiess, 1984). Souvent, les services utilisent les mêmes liens physiques. Donc, l'affichage de plusieurs services sur une même image nécessite un mécanisme assez complexe qui décale les services pour que chacun puisse être suivi individuellement.

Contrairement à l'affichage des lignes de transport en commun, nous devons aussi identifier les sous-chemins entre les gares où les services arrêtent. Dans ce but, nous utilisons des symboles sur les noeuds. La figure 7.17 illustre la technique que nous employons. Les étoiles représentent les gares d'arrêt et les petits carrés montrent les noeuds intermédiaires.

Une représentation graphique des chemins devient rapidement incompréhensible quand le nombre de services affichés sur un même écran est trop élevé. Afin de pallier à cette lacune, nous offrons à l'utilisateur la possibilité de diviser l'écran en zones et d'afficher un service par zone.

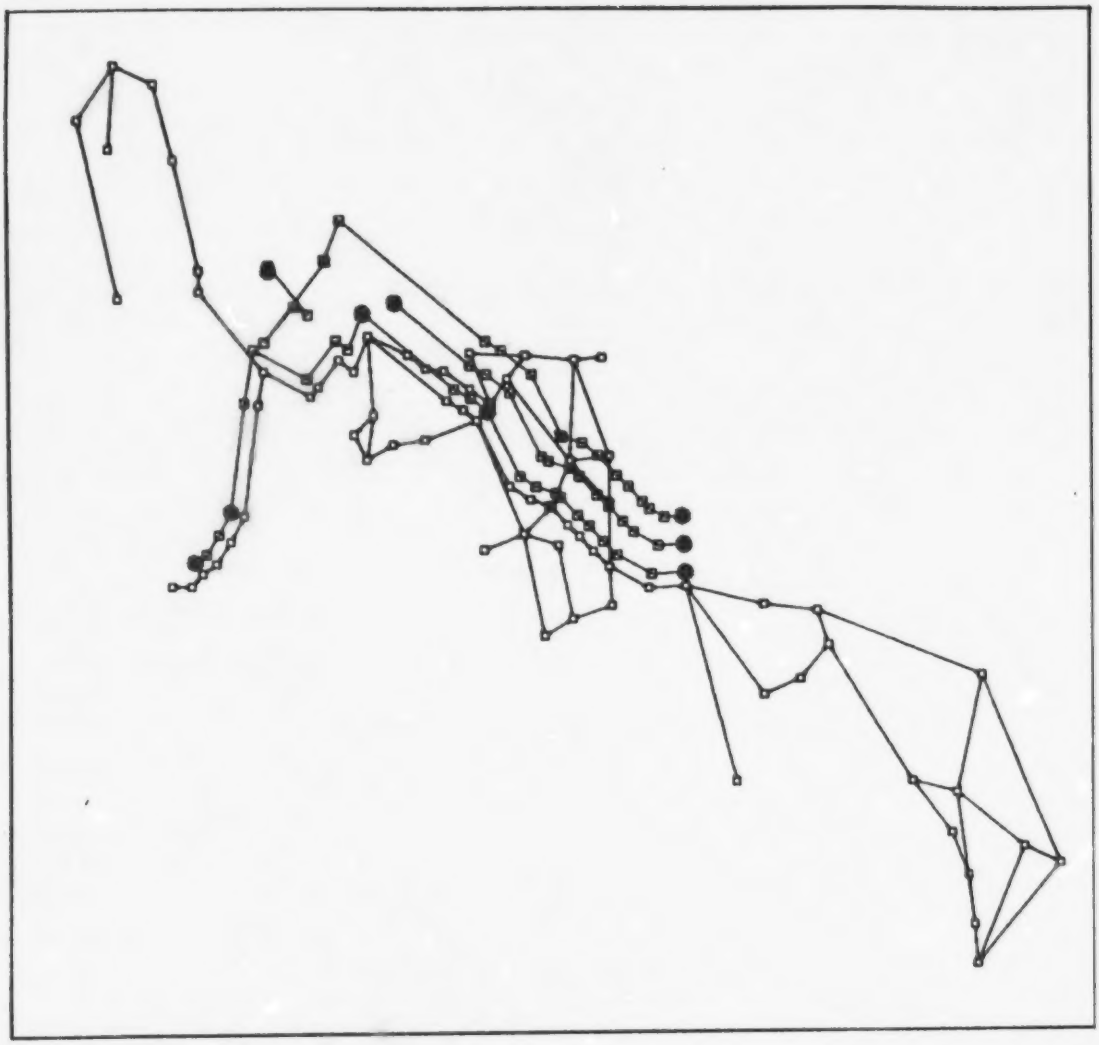


Figure 7.17 : Quelques services sur le réseau physique

Les figures 7.18 et 7.19 montrent que l'utilisateur est responsable de la composition de l'image. Ceci, lui permet de maximiser l'utilisation de la surface de visualisation. Mentionnons que cette approche est moins efficace lorsque le terminal graphique ne permet pas l'effaçage sélectif.

Nous terminons cette section en présentant d'autres sorties graphiques pour la représentation des chemins.

La figure 7.20 illustre le flux total de marchandises sur les arcs d'un service. Les noeuds origine et destination sont représentés par des cercles, les noeuds où le service arrête sont identifiés par des rectangles bleu foncé tandis que les noeuds des sous-chemins sont des rentangles bleu pâle. Nous avons dessiné les autres noeuds du réseau physique par des petits carrés. Ainsi, l'emphase est mise sur la description du service. L'analyse détaillée du service est contenue dans une boîte, à droite de l'image située au noeud origine du service. Notons que la position de cette boîte est choisie par l'utilisateur. De plus, dans le coin inférieur gauche de l'écran, en plus d'indiquer la portion de la fenêtre qui est reproduite dans la partie centrale du dessin, nous avons identifié les noeuds origine et destination du service avec deux étoiles.

À la figure 7.21, nous montrons l'affichage de quatre itinéraires parallèles desservant un marché. La description du marché est contenue dans une boîte que nous avons placée au-dessus du noeud origine du marché. Les chaînes de caractères dessinées sur les flèches indiquent le service emprunté par l'itinéraire sur ce lien. Rappelons que les arcs d'itinéraires sont composés de sous-chemins dans le réseau de service reliant deux terminus (rectangles rouges) où la marchandise est classifiée. Le rectangle bleu indique qu'à cet endroit, l'arc d'itinéraire emprunte un nouveau service. Comme précédemment, nous indiquons dans le coin inférieur gauche du dessin, la portion de la fenêtre qui est reproduite dans la partie centrale de l'écran. Ainsi, nous contatons que la description des itinéraires est incomplète car une des étoiles est en dehors du rectangle bleu.

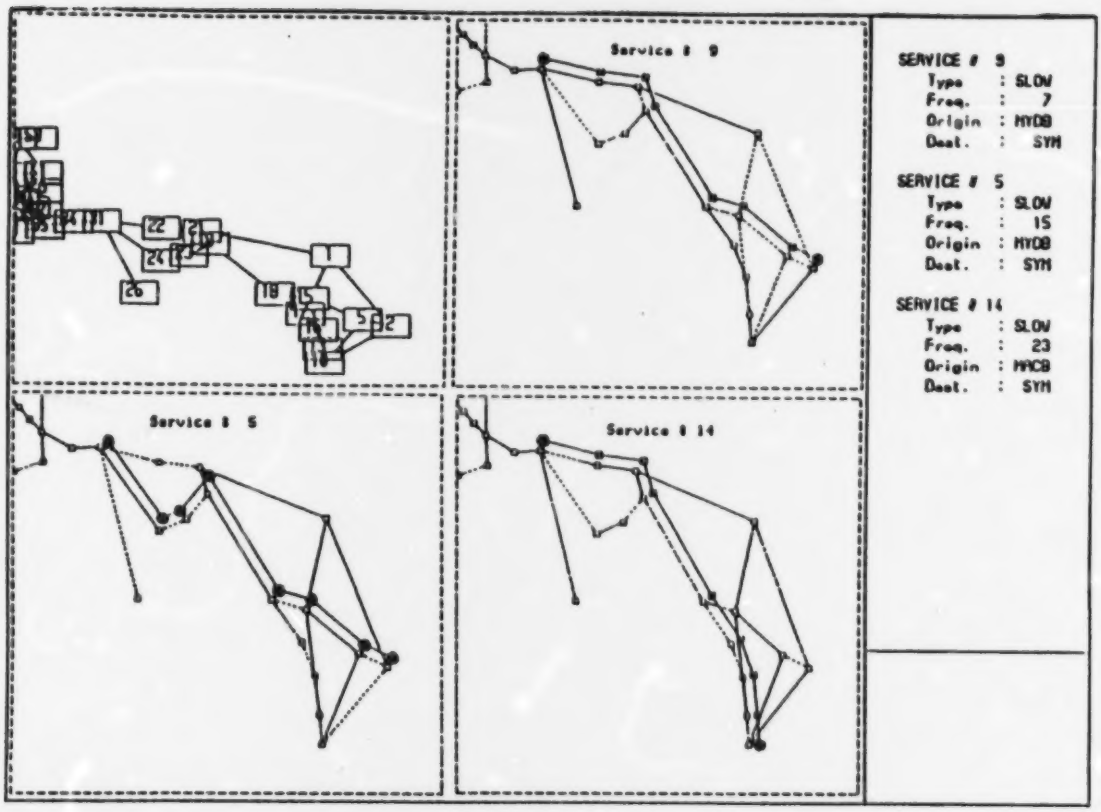


Figure 7.18 : Quelques services sur un même écran

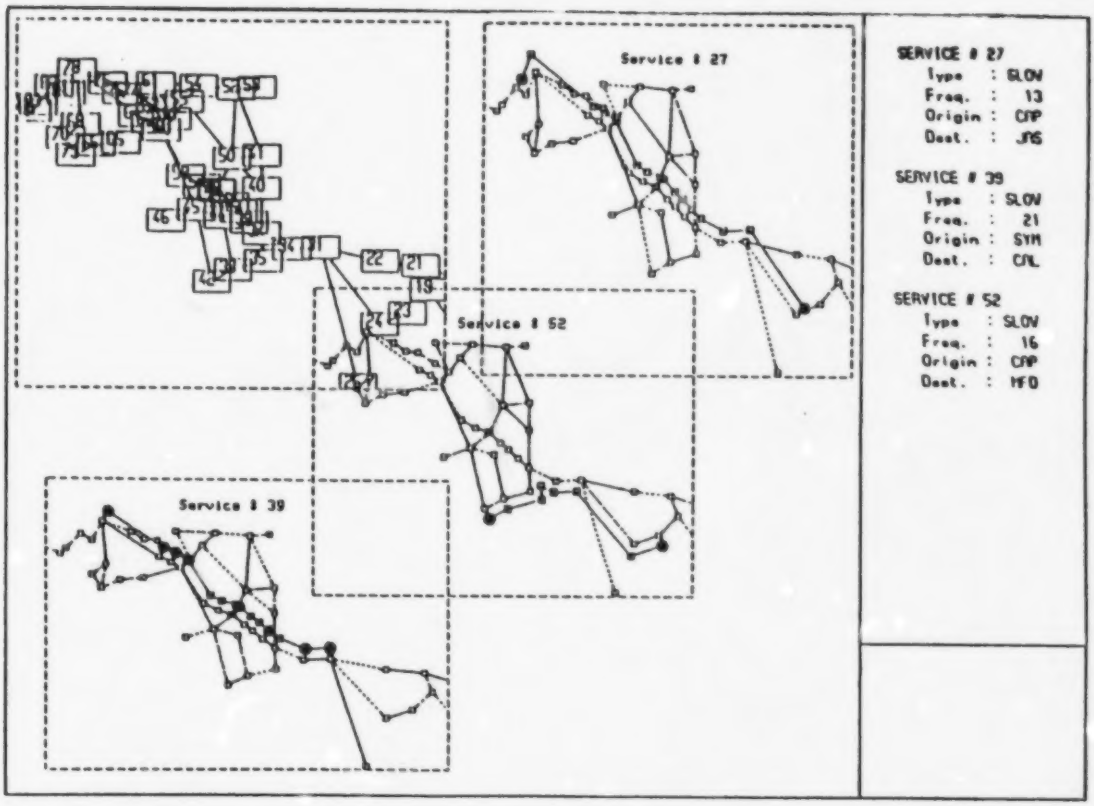


Figure 7.19 : "Mauvaise" composition de l'image sur un terminal qui ne permet pas l'effaçage sélectif

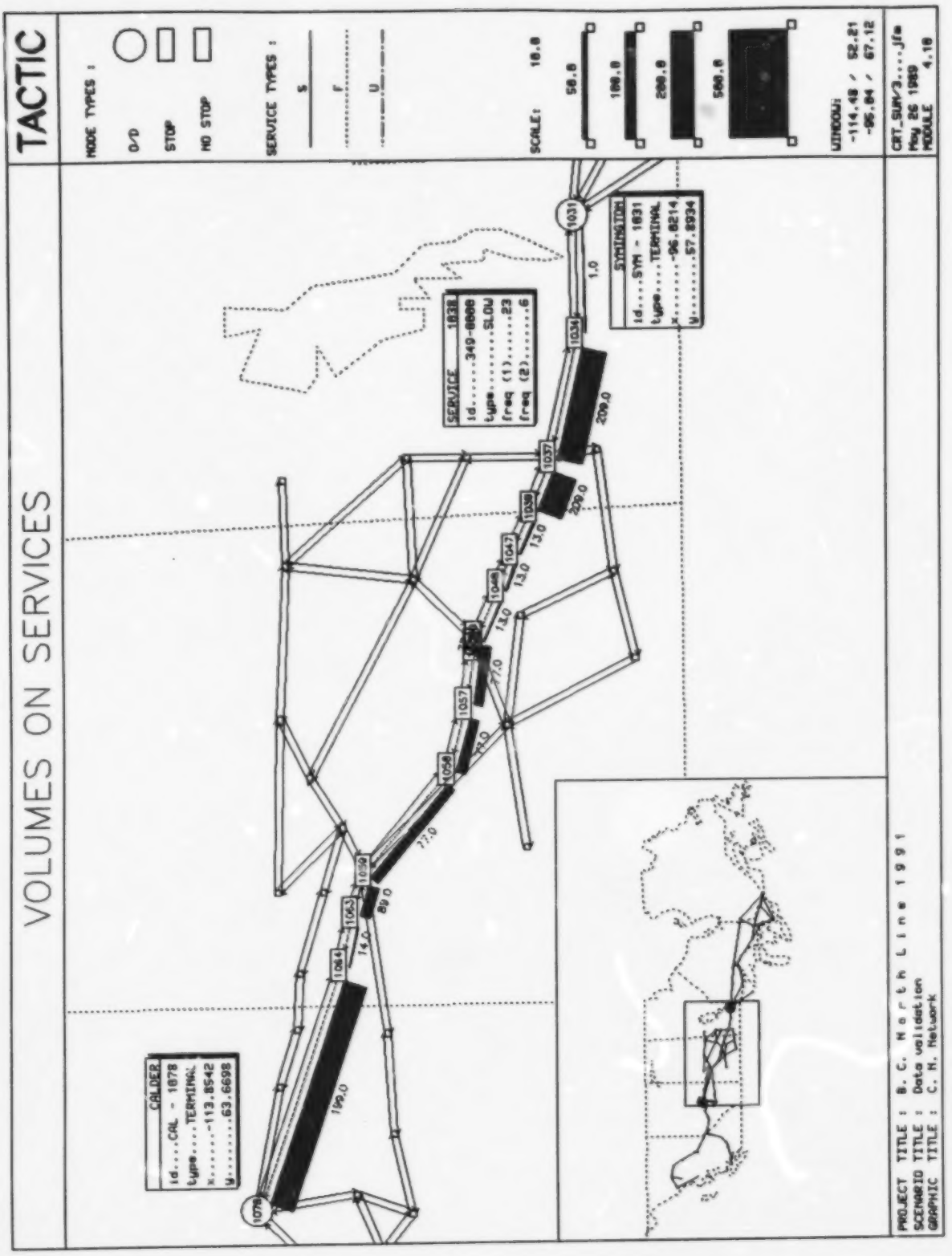


Figure 7.20 : Flux de marchandises sur les arcs d'un service

Figure 7.21 : Un marché et quatre itinéraires parallèles

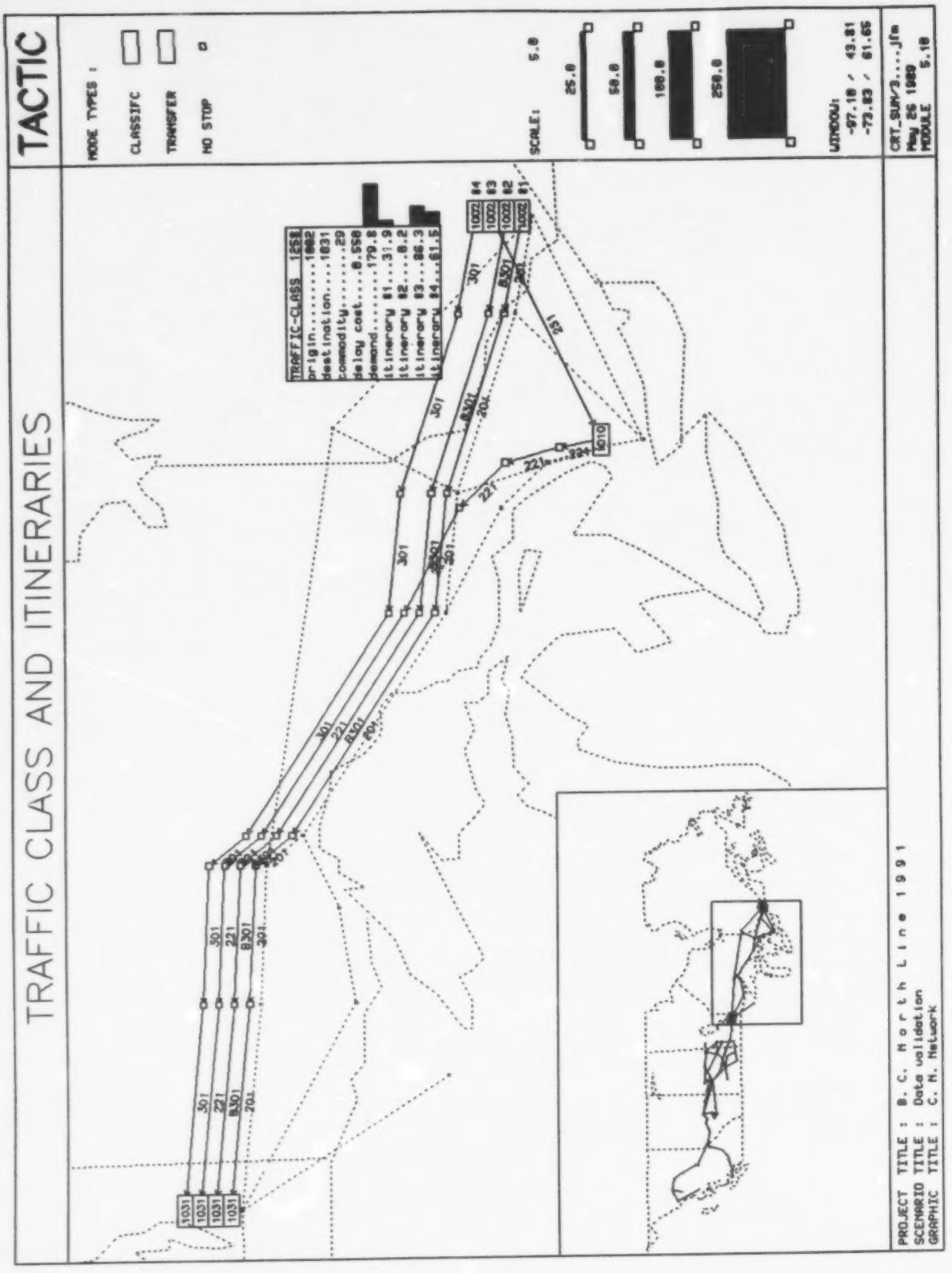


Figure 7.22 : Un marché et quatre itinéraires non parallèles

La figure 7.22 illustre un autre marché desservi par quatre itinéraires. Cette fois-ci la description est complète. Remarquons que le troisième itinéraire n'est pas parallèle aux autres.

7.8 TACTIC du point de vue d'un usager

Comme nous l'avons mentionné plus tôt, TACTIC est un ensemble d'outils interactifs graphiques pour l'aide à la planification tactique et stratégique des systèmes de transport des marchandises. Les outils sont des programmes indépendants, que nous appelons **modules**. Tous les modules ont accès à une même base de données. Chaque module accomplit une tâche spécifique permettant au planificateur de se concentrer sur un aspect précis du problème. Rappelons que la structure des modules a été décrite à la section 7.1. Notre but n'est pas de fournir ici un manuel d'usager; le système informatique n'en étant qu'à ses premières phases de développement. Ainsi, beaucoup de modifications risquent encore d'être apportées. Dans cette section nous voulons plutôt donner un aperçu des modules réalisés jusqu'à maintenant ainsi que leur fonctionnement et de ceux que nous prévoyons développer dans une phase ultérieure du projet.

Chaque module est identifié par un numéro de la forme X.YY où X indique le groupe auquel le module appartient, et YY le distingue à l'intérieur de ce groupe. Pour TACTIC, nous avons identifié les six groupes de modules suivants :

.1 Données de base
.2 Analyse des résultats
.3 Éditeur de fonctions
.4 Éditeur des demandes de transport
.5 Procédures d'optimisation
.6 Utilitaires

Notons que le numéro associé à chacun des groupes est donné à titre indicatif seulement. Cela peut changer.

Passons en revue, un à un, chacun des groupes de modules.

1) DONNÉES DE BASE

Les modules de ce groupe permettent de gérer l'information concernant le réseau physique (noeuds et liens) et la structure des services (services, arcs de service et routes). Pour tous ces éléments, l'entrée et la modification des données peuvent être faites à l'aide de fichiers séquentiels préparés à l'avance, ou d'une façon interactive. Les modifications interactives-graphiques sont faites avec des feuilles de travail graphiques (voir section 7.2). Quant à la sortie, l'usager peut générer, pour chaque type d'élément, des rapports alphanumériques, destinés soit à l'imprimante sous forme de fichiers séquentiels soit au terminal sous forme de chaînes de caractères ou de dessin.

Actuellement, nous avons programmé, en langage C, un seul module de ce groupe. C'est le module 1.10 (voir figures 7.2 et 7.8) qui permet l'inspection interactive-graphique des caractéristiques du réseau physique. Aussi, mentionnons que nous avons expérimenté l'affichage des services (voir figures 7.17, 7.18 et 7.19) en FORTRAN.

De plus, nous prévoyons ajouter des modules permettant le calcul et la visualisation des chemins les plus courts sur le réseau physique et sur le réseau de service. Ces outils servent aussi bien à la validation des données du réseau (e.g. la connexité) qu'à répondre à plusieurs questions intéressantes reliées aux chemins et distances sur le réseau physique et le réseau de service.

Enfin, nous pensons aussi doter TACTIC d'un module pour la génération des services. Soulignons que, présentement, le réseau de service doit être fourni par le planificateur et, l'input de ce réseau influence directement la qualité de la solution. Ainsi, il est intéressant de fournir au

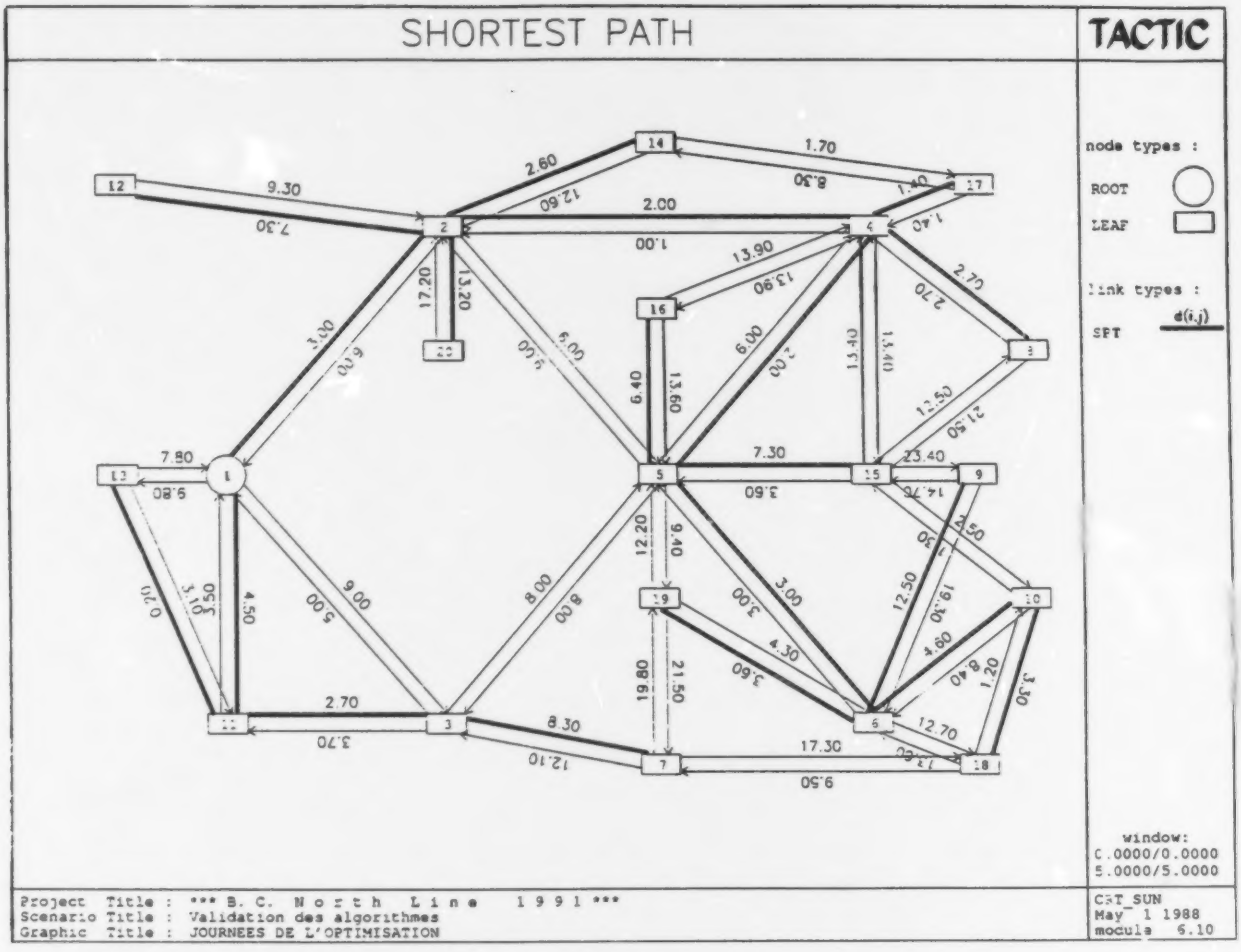


Figure 7.23 : Représentation des chemins les plus courts

planificateur un moyen de générer des services à l'aide d'algorithmes de recherche opérationnelle tels les algorithmes de cheminement dans les réseaux (voir figure 7.23).

2) ANALYSE DES RÉSULTATS

Le groupe 2 comprend les modules qui permettent au planificateur d'analyser les activités sur les liens du réseau physique, les activités aux noeuds, les activités sur les services et le cheminement des marchandises. Ces résultats ou activités proviennent soit d'une situation réelle (ou observée) ou des résultats d'un algorithme de recherche opérationnelle.

- Analyse sur les liens du réseau physique : Grâce à ces modules, le planificateur peut examiner les activités sur les liens du réseau physique, telles que listées à la section 7.7.1, item B.1. Il peut visualiser, soit le flux total de marchandises sur les liens, soit une décomposition de ce flux pour un même attribut (e.g. flux total de marchandises par type de commodité).

  Présentement, nous pouvons mener cette analyse à l'aide du module 2.10 (voir figure 7.15). Ce programme affiche le réseau physique avec le flux total de marchandises sur les liens, et à l'aide de commandes (propres à ce module) le planificateur peut obtenir des histogrammes montrant la répartition de ce flux pour l'attribut qu'il a spécifié. Nous prévoyons aussi implanter en langage C, la méthode d'affichage illustrée à la figure 7.14.

- Activités aux noeuds : Ces modules sont semblables à ceux décrits au paragraphe précédent à la seule différence que les résultats montrés apparaissent sur les noeuds. Nous avons énuméré les données importantes à analyser à l'item B.2 de la section 7.7.1.

Les figures 7.11 et 7.16 sont des sorties graphiques obtenues avec un module d'analyse des activités aux noeuds. Nous prévoyons aussi implanter la méthode d'affichage illustrée à la figure 7.14 en remplaçant les liens par des noeuds.

- Activités sur les services : Ce sous-groupe comprend les modules qui permettent l'examen des résultats sur la structure des services. Pour l'instant, nous avons un programme (voir figure 7.20) qui affiche un seul service sur le réseau physique avec le flux total de marchandises sur les arcs. Comme pour les modules précédents, nous mettrons au point des commandes pour décomposer le flux selon un attribut en utilisant des histogrammes ou en partitionnant la surface de visualisation. Nous désirons aussi représenter les arcs de service d'une manière conceptuelle, c'est-à-dire en ignorant les routes dans le réseau physique (voir figure 7.24).

- Marchés et itinéraires : Finalement, ces derniers modules rendent possible la visualisation de l'acheminement des marchandises pour chacun des marchés. Jusqu'à présent, nous avons implanté un module qui affiche tous les itinéraires (sur le réseau physique) qui desservent un marché (voir figures 7.21 et 7.22). Mentionnons que pour ces programmes, il n'est pas utile d'afficher le flux sur les arcs d'un itinéraire car il est le même sur tous les arcs. Aussi, nous songeons à représenter les itinéraires sur le réseau de service conceptuel.

3) ÉDITEUR DE FONCTIONS

Un aspect important de notre modèle de planification est sa capacité de s'adapter à plusieurs contextes. Ainsi, le planificateur a la possibilité de définir ses propres fonctions de coût et de délai. Avec l'outil

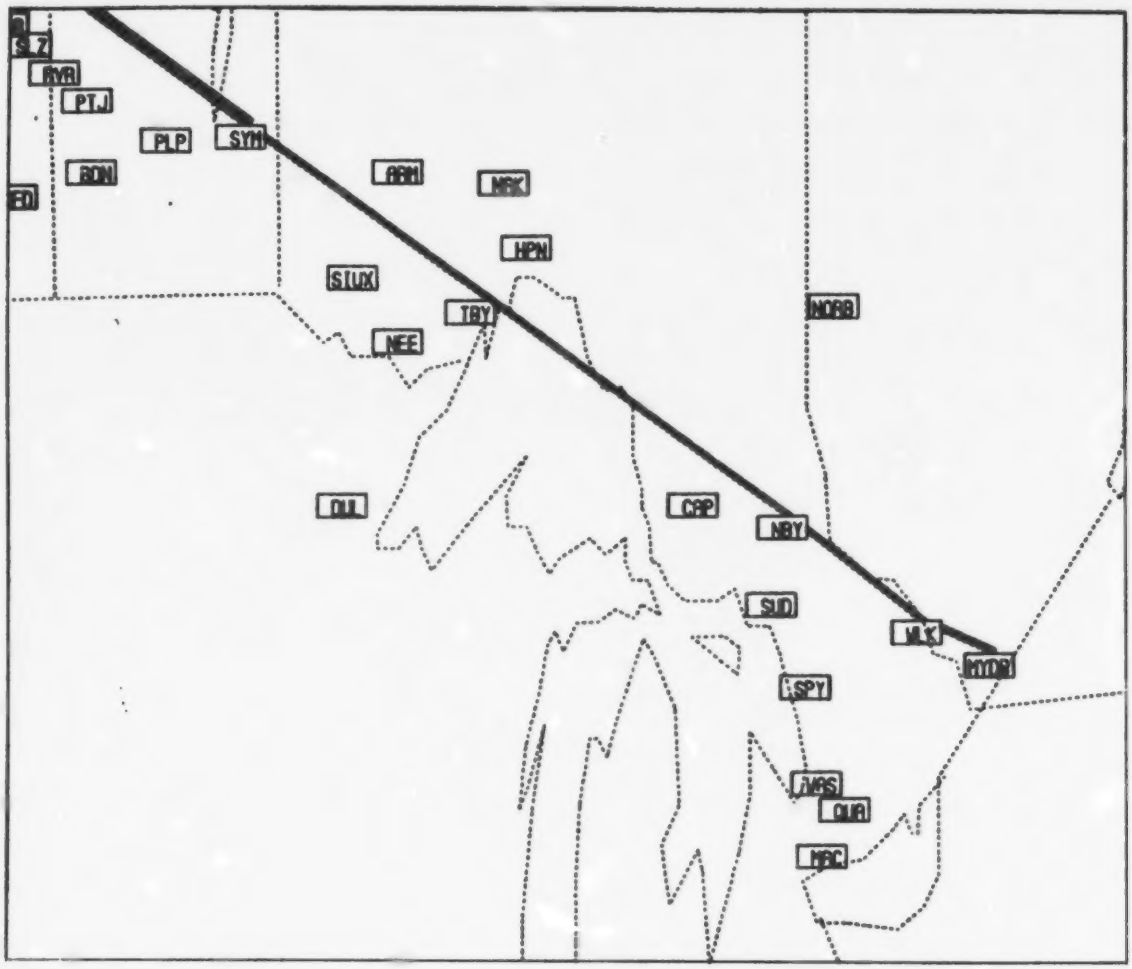


Figure 7.24 : Représentation conceptuelle d'un service

TACTIC cela peut se faire à l'aide de tables de fonctions prédéfinies ou à l'aide d'expressions algébriques entrées par l'usager. Pour l'instant, nous n'avons pas traité ce sujet. C'est une étape ultérieure au projet.

4) ÉDITEUR DES DEMANDES DE TRANSPORT

Dans une phase future de ce projet, nous prévoyons doter l'outil TACTIC de programmes pour la modification interactive des demandes de transport des marchandises.

5) PROCÉDURE D'OPTIMISATION

Ce groupe comprend tous les programmes du noyau méthodologique pour la résolution du problème de la planification tactique-stratégique des systèmes de transport. Acutellement, ces programmes, de type heuristique, se trouvent sur l'ordinateur Cyber 835/855 de l'Université de Montréal. Le langage de programmation est FORTRAN. Une prochaine étape du projet consiste à implanter ces programmes en langage C et les incorporer dans l'outil TACTIC. Aussi, des développements récents, tant en optimisation (tels les nouveaux algorithmes de relaxation-primal/dual, ou d'hyperplans de coupe), qu'en informatique (calcul parallèle, par exemple) nous incitent à aborder la mise au point d'algorithmes exacts.

Toutefois, aucun modèle ne peut tenir compte de tous les cas possibles rencontrés dans la réalité. Pour cette raison, nous désirons inclure des procédures avec lesquelles le planificateur peut incorporer des règles et des critères non inclus dans le modèle, telle la préspécification de certains cheminements ou des limitations sur le nombre de blocs ou de trains à former à un terminus donné. Nous prévoyons aussi prendre avantage de la modularité des algorithmes développés jusqu'à maintenant et inclure des procédures de post-optimisation afin d'optimiser soit le design du réseau de service, soit la distribution du traffic. Ainsi, le planificateur devient maître du déroulement même du processus d'optimisation en ayant la possibilité d'intervenir entre chaque phase de l'algorithme.

C'est aussi par des modules de ce groupe que l'utilisateur spécifie les fonctions de coût et de délai à être employées.

6) UTILITAIRES

Finalement, les modules de ce groupe sont liés à la gestion de l'environnement de travail et sont relativement indépendants du problème de planification. Pour cette raison, nous pensons implanter les modules utilitaires semblables à ceux du logiciel STAN (Crainic, 1989) dans TACTIC.

Les fonctions principales de ces modules permettent des manipulations directes de la banque de données, la consultation d'un journal de bord relatant l'histoire récente de la banque de données ainsi que l'entrée et la modification des lignes de démarcation.

Nous prévoyons aussi mettre au point un module pour permettre à un usager d'établir les unités de mesure (kilomètre ou mille, tonne ou nombre de véhicules, etc.) avec lesquelles il désire travailler et les titres principaux qui apparaissent sur les rapports et les graphiques. De plus, il sera possible de redéfinir des termes tels que TERMINAL, STATION, JUNCTION, SIMPLE et DOUBLE, etc., qui apparaissent dans les légendes pour les remplacer par des termes avec lesquels le planificateur est plus familier. Cette particularité permet d'adapter l'outil à chaque application ("customize") et non l'inverse. Ceci, nous l'espérons, contribuera à une plus grande diffusion de celui-ci.

Nous terminons ce chapitre en discutant des commandes graphiques que nous avons mises au point jusqu'à maintenant.

Les **commandes graphiques** servent à modifier et manipuler le dessin, une fois qu'il est affiché à l'écran. Ces commandes sont les mêmes pour chacun des modules d'affichage graphique.

Un aspect important des systèmes interactifs-graphiques est la façon dont s'effectue la communication entre le système et l'utilisateur. De plus, il est important de respecter le degré d'expérience de l'usager avec l'outil.

Nos premiers programmes graphiques (Crainic et Mondou, 1986; voir figure 7.14) utilisaient un menu affiché au bas de l'écran. Pour activer une commande, nous pointons avec un dispositif d'entrée graphique la case qui correspond à la commande désirée. Nous avons constaté que cette méthode rend difficile l'ajout de nouvelles commandes (nouveau design du menu). De plus, cette façon d'entrée les commandes est fastidieuse à la longue et elle a pour conséquence qu'une seule directive peut être exécutée à la fois. pour ces raisons, nous adoptons le concept utilisé dans plusieurs autres systèmes interactifs-graphiques (STAN : Crainic, 1989; PLANAIR : Lebeuf, 1985; EMME/2 : Spiess, 1984) qui consiste à invoquer une commande par l'entrée d'un seul caractère. De plus, si l'équipement le permet, ce caractère n'apparaît même pas au terminal afin de ne pas gâter le contenu de l'écran. Cependant, nous prévoyons doter les programmes graphiques d'une commande demandant l'affichage d'un menu dans le cas où un utilisateur novice aurait un blanc de mémoire.

Il y a deux types de commandes graphiques; celles nécessitant le réaffichage du dessin et les autres. Les commandes du premier type sont aussi appelées des commandes qui causent un arrêt. Les commandes de ce type sont en lettres majuscules pour les distinguer des autres commandes. En voici une brève description :

"W" : ("change window") Cette directive permet à l'utilisateur d'effectuer un changement de la portion du réseau qui est affichée dans la partie principale de l'écran. Des sous-commandes permettent la spécification de la nouvelle fenêtre selon une des méthodes suivantes :

"c" : la nouvelle fenêtre est spécifiée à l'aide du curseur graphique. L'usager pointe sur les coins inférieur gauche et supérieur droit de la fenêtre désirés.

"p" : à l'aide du curseur graphique, l'usager indique le point sur le réseau qui constitue le centre de la nouvelle fenêtre ("pan"). Les dimensions de la fenêtre demeurent inchangées.

"r","l","u","d" : la nouvelle fenêtre est adjacente à la fenêtre actuelle. Cette option permet de se "promener" dans le réseau en affichant la fenêtre voisine à droite ("right"), à gauche ("left"), en haut ("up") ou en bas ("down").

"z" : la nouvelle fenêtre est obtenue en agrandissant ou rapetissant la fenêtre actuelle par un certain facteur ("zoom").

"n" : la nouvelle fenêtre est spécifiée sur le squelette du réseau ("network") en pointant les coins inférieur gauche et supérieur droit de la fenêtre désirée (voir figure 7.8, section 7.7.2).

"^" : le système garde une pile des dix fenêtres qui ont été utilisées le plus récemment. Cette sous-commande permet de remonter à la dernière fenêtre.

"S" : ("change Scale") L'usager demande un changement de l'échelle des attributs.

"H" : ("change Hatch") L'usager demande une nouvelle valeur pour l'espacement entre les vecteurs de remplissage.

"T" : ("change Title") L'usager demande un nouveau titre pour le dessin (voir section 7.2).

"?" : ("help") L'utilisateur a un blanc de mémoire. (Cette fonction n'est pas encore implantée.)

"Q" : ("Quit") L'exécution du module courant est terminée et le contrôle est passé au processeur principal.

"R" : ("Restart") Cette directive permet de recommencer au début du module courant. L'utilisateur peut ainsi remettre les valeurs par défaut aux paramètres.

Finalement, voici les commandes qui ne nécessitent pas un réaffichage de l'écran :

"1" ... "9" : L'ensemble des lignes de démarcation (1 à 9) est superposé au dessin.

"h" : ("hard copy") Cette directive permet d'envoyer le contenu de l'écran à l'unité d'impression relié au terminal (e.g. imprimante Tektronix 4696 attaché au terminal Tektronix 4115). Cette commande ne produit rien s'il n'y a aucune unité reliée au terminal.

"p" : ("send to plot file") Le dessin courant est regénéré et redirigé vers le fichier des dessins pour utilisation ultérieure.

"w" : ("where am I" ou "show current window") Cette commande dessine dans le coin inférieur gauche de la partie centrale de l'écran le squelette du réseau et un rectangle indiquant la portion du réseau qui est présentement dessinée.

CONCLUSION

Ce mémoire représente un pas important pour la mise en oeuvre d'un système interactif-graphique pour la planification tactique du transport interurbain des marchandises.

Rappelons que les problèmes du chemin le plus court, faisant partie du modèle mathématique que nous utilisons actuellement, représentent environ 35 % du temps total d'exécution de l'algorithme. Ainsi, notre contribution à la partie méthodologique consiste à l'analyse détaillée des algorithmes du chemin le plus court. Nous avons étudié leurs performances par rapport à SHEAP, celui qui est présentement utilisé pour l'optimisation de l'acheminement des marchandises à travers le réseau de service. Nos expérimentations numériques ont été menées en langage C en prenant soin de prendre avantage des caractéristiques de ce langage. Nos résultats montrent que LTHRS est la procédure la plus efficiente pour tous les réseaux que nous avons testés. De plus, nous avons discuté des impacts de la programmation en langage C sur l'efficacité de ces algorithmes. À cet effet, soulignons que l'usage des pointeurs permet une économie d'au moins 15 % du temps d'exécution des algorithmes du chemin le plus court par rapport aux implantations usant des tableaux et des faux-pointeurs. Cela est certainement plus grand si nous comparons ces implantations avec des programmes écrits en FORTRAN ou en PASCAL. De plus, nous avons observé des gains allant jusqu'à 30 % dans certains cas. Aussi, nous remarquons que pour la plupart des algorithmes, les implantations usant des pointeurs ne sont pas plus compliqués que les mises en oeuvre traditionnelles.

À notre connaissance, il n'existe aucun résultat d'analyse numérique pour l'algorithme SRHEAP. Dans ce mémoire, nous avons pallié à cette lacune en rapportant plusieurs résultats expérimentaux le concernant. Bien que cet

algorithme possède une très bonne borne théorique, en pratique, pour les réseaux peu denses, il est dominé par les autres méthodes qui procèdent par ajustements sélectifs. Toutefois, pour les réseaux complets, il est aussi rapide que les autres.

Enfin, nous avons aussi examiné les algorithmes de réoptimisation pour la détermination des chemins les plus courts entre tous les sommets d'un graphe. Cependant, nos résultats numériques sont décevants et ils contredisent ceux présentés dans la littérature scientifique. Cela provient du fait que l'ordre dans lequel sont visités les sommets pour le calcul des arborescences influence l'efficacité des procédures de réoptimisation. Ainsi, dans le cas des applications réelles, les auteurs mentionnent qu'une analyse du réseau, a priori, a été conduite afin de déterminer la séquence de visite des sommets. Pour notre part, nous avons simplement pris les sommets dans l'ordre lexicographique. Le problème consiste donc à déterminer une bonne séquence et, éventuellement, la meilleure. Bien que ce travail soit très intéressant, il dépasse largement les cadres de ce mémoire. Pour cette raison, nous n'avons pas exploré en profondeur ces avenues de recherche. Pour ce problème, il est possible d'adapter des heuristiques pour le problème du commis-voyageur afin de déterminer une bonne séquence a priori.

Parallèlement, une partie importante de ce projet de recherche fut consacrée à l'étude des problèmes particuliers de représentation et d'affichage des éléments intervenant lors du processus de planification des opérations de transport des marchandises.

Malgré le fait que nous disposions déjà d'une bibliothèque de fonctions graphiques écrites en FORTRAN, nous avons opté de mettre en oeuvre l'outil TACTIC en langage C. Nous avons choisi ce langage principalement à cause de l'efficacité de son code, nos expérimentations numériques des algorithmes du chemin le plus court le confirmant. De plus, nous

croyons que le langage C peut contribuer à rendre l'outil TACTIC facilement transportable d'un environnement informatique à un autre. Pour faire suite au choix de ce langage, nous avons implanté une nouvelle bibliothèque de fonctions graphiques, adaptée au graphisme de réseau, en langage C. Jusqu'à maintenant, celle-ci a été employée avec succès sur trois environnements différents.

Danc ce mémoire, nous avons discuté des solutions que nous apportons aux difficultés de représentation et d'affichage des éléments du modèle. Nous avons pris soin de mettre en oeuvre des solutions générales à ces problèmes de manière à ce qu'elles puissent être facilement adaptées à d'autres problèmes connexes. Notre travail n'est pas seulement une aventure académique puisque nos expérimentations ont été menées à l'aide de données provenant d'une application actuelle au transport des marchandises par chemin de fer.

Enfin, nous avons analysé et implanté des structures de données favorables à l'exécution des algorithmes ainsi qu'à l'accès rapide des informations pour les manipulations graphiques. Soulignons que pour cette tâche, le choix du langage C a été judicieux. Aussi, nous avons développé les premières interfaces de communication avec l'usager tels le langage des commandes et la structure des dialogues.

Beaucoup de travail reste à faire avant d'en arriver à un logiciel complet, interactif-graphique d'aide à la planification tactique du transport interurbain des marchandises. Toutefois, dans ce mémoire, nous avons établi les conditions pour qu'un tel logiciel soit opérationnel et nous avons créé les fondations pour les travaux futurs.

L'étude du noyau méthodologique est presque complétée. Actuellement, les programmes d'optimisation, de type heuristique, se trouvent sur l'ordinateur Cyber de l'Université de Montréal. Le langage de programmation est

FORTRAN. Une prochaine étape du projet consiste à implanter ces programmes en langage C et les incorporer dans l'outil TACTIC. Aussi, des développements récents, tant en optimisation (tels les nouveaux algorithmes de relaxation primal-dual, ou d'hyperplans de coupe), qu'en informatique (calcul parallèle, par exemple) nous incitent à aborder la mise au point d'algorithmes exacts.

Du point de vue de la mise en oeuvre informatique de l'outil TACTIC, le problème le plus difficile est celui de l'organisation physique des données dans la banque. Rappelons que la dimension "chemin" est particulière à notre logiciel. Pour cette raison, nous ne pouvons pas facilement employer des principes de développement des banques de données employés dans d'autres systèmes interactifs-graphiques adaptés aux problèmes de réseau. Jusqu'à maintenant, nous avons envisagé deux solutions et nous poursuivons nos travaux en comparant leurs avantages et désavantages. De plus, nous comptons faire des lectures plus poussées sur ce sujet.

À plus long terme, nous prévoyons que TACTIC pourra être utilisé lors du processus de planification des opérations connexes au transport des marchandises. Nous pensons plus particulièrement au transport à vide et à la planification des systèmes de distribution des entreprises industrielles. De plus, dans les cas où il ne sera plus avantageux d'utiliser un noyau méthodologique différent de celui que nous développons, il sera quand même possible d'employer les procédures graphiques et des principes de base que nous avons présentés dans ce mémoire.

BIBLIOGRAPHIE

Aho, A., Hopcroft, J., Ullman, J., Structures de données et Algorithmes, Addison-Wesley Europe, Inter Editions, Paris, 1987.

Ahuja, R.K., Mehlhorn, K., Orlin, J.B., Tarjan, R.E, *Faster Algorithms for the Shortest Path Problem,* Technical report No. 193, Operations Research Center, M.I.T., April 1988.

Andreasson, I., *The Volvo Approach to Computer-Aided Transit Planning,* Transportation Research Record 657, 1978.

Babin, A., Florian, M., James-Lefebvre, L., Spiess, H., *EMME/2: An Interactive Graphic Method for Road and Transit Planning,* Transportation Research Record 866(1-9), 1982.

Bellman, R., *On a Routing Problem,* Quart. Appl. Math. 16, 1958.

Bourachot, F., *Conception assistée par ordinateur de l'exploitation et de l'aménagement des gares ferroviaires voyageurs : modèle intercatif graphique AFAIG,* thèse de doctorat, Ecole Polytechnique Fédérale de Lausanne, Suisse, 1984.

Bovet, J., *Une amélioration de la méthode de Dijkstra pour la recherche d'un plus court chemin dans un réseau,* Discrete Applied Mathematics 13, 1986.

Bradley, G.H., Brown, G.G., Graves, G.W., *Design and Implementation of Large Scale Primal Transhipment Algorithms,* Management Science 24(1), 1-34, 1977.

Crainic, T.G., *Un modèle de planificatin tactique pour le transport ferroviaire des marchandises,* thèse de doctorat, dép. d'informatique et de recherche opérationnelle, Université de Montréal, 1982.

Crainic, T.G., *Tactical Planning: Issues, Models and Tools",* publication #520, Centre de recherche sur les transports, Université de Montréal, 1987.

Crainic, T.G., *Rail Tactical Planning: Issues, Models and Tools, in Freight Transport Planning and Logistics,* L. Bianco and A. La Bella (Editors), 463-509, Springer-Verlag, 1989.

Crainic, T.G., *An Interactive-Graphic System for the Analysis and Evaluation of Regional/National Transportation Networks,* publication #632, Centre de recherche sur les transports, Université de Montréal, 1989.

Crainic, T.G., Rousseau, J.-M., *Multicommodity, Multimode Freight Transportation: A General Modeling and Algorithmic Framework for the Service Network Design Problem,* Transportation Research B 20B(3), 1986.

Crainic, T.G., Roy, J., *O.R. Tools for Tactical Freight Transportation Planning,* publication #398, Centre de recherche sur les transports, Université de Montréal, 1985.

Crainic, T.G., Mondou, J.-F., *An Interactive Graphic Planning System for Freight Transportation Networks*, ORSA/TIMS Joint national meeting, Miami, 27-29 octobre 1986.

Crainic, T.G., Mondou, J.-F., *Systèmes interactifs graphiques pour la planification des réseaux de transport interurbain des marchandises - Méthodologie et premiers résultats*, publication #467, Centre de recherche sur les transports, Université de Montréal, 1986.

Crainic, T.G., Nicolle, M.-C., *Planification tactique du transport ferroviaire de marchandises : quelques aspects de modélisation*, publication #460, Centre de recherche sur les transports, Université de Montréal, 1986.

Crainic, T.G., Dejax, P., *The Use of Computer Interactive-Graphic Systems in the Modelling of Distribution and Transportation Systems*, INFORS '87, Buenos Aires, 1987.

Crainic, T.G., Mondou, J.-F., *TACTIC: Recents Developments*, CORS/TIMS/ORSA Joint National Meeting, Vancouver, mai 1989.

Crainic, T.G., Mondou, J.-F., *TACTIC: A Computer-Aided Planning (CAP) System for Rail Networks and Operations at Tactical and Strategic Levels*, Centre de recherche sur les transports, Université de Montréal, 1989.

Crainic, T.G., Mondou, J.-F., *Interactive-Graphic Interfaces for Planning the Transportation and Distribution of Freight*, Centre de recherche sur les transports, Université de Montréal, 1989.

Crainic, T.G., Roy, J. *OR Tools for the Tactical Planning of Freight Transportation*, European Journal of Operations Research 33(3), 1988.

Dantzig, G.B., *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J., 1963.

Delorme, L., Crainic, T.G., Roy, J., *Une nouvelle approche pour la planification des opérations de transport des marchandises*, publication #444, Centre de recherche sur les transports, Université de Montréal, 1986.

Delorme, L., Roy, J., Rousseau, J.-M., *Motor-Carriers Operations Planning Models*, Proceedings of the 2nd International Seminar on Freight Transport Planning and Logistics, Bressanone, Italie, juillet 1987.

Denardo, E.V., Fox, B.L., *Shortest-Route Methods: 1. Reaching, Pruning, and Buckets*, Operations Research 27(1), 1979.

Dial, R.B., *Algorithm 360: Shortest Path Forest with Topological Ordering*, Commun. A.C.M. 12(11), 1969.

Dial, B., Glover, F., Karney, D., Klingman, D., *A Computational Analysis of Alternatives Algorithms and Labeling Techniques for Finding Shortest Path Trees*, Networks 9, 215-248, 1979.

Dijkstra, E.W., *A Note on Two Problems in Connexion with Graphs*, Numerishe Mathematik 1, 1959.

Divoky, J.J., *Improvements for the THRESH X2 Shortest Path Algorithm*, Operations Research Letters 6(5), 1987.

Dreyfus, S.E., *An Appraisal of Some Shortest Path Algorithms,* Operations Research 17, 395-412, 1969.

Fisher, M.L., *Interactive Optimization,* Annals of Operations Research 5, 541-556, 1985/1986.

Florian, M., Nguyen, S., Pallotino, S., *A Dual Simplex Algorithm for Finding all Shortest Paths,* Networks 11, 367-378, 1981.

Florian, M., Senécal, L., Nguyen, S., Crainic, T.G., *A Multi-Product, Multi-Mode, Freight Assignment Model for National Transportation Planning,* TIMS/ORSA Joint National Meeting, Los Angeles, 1986.

Florian, M., Crainic, T.G., Guélat, J., Babin, A., *STAN: An Interactive-Graphic System for Strategic Transportation Analysis at the National and Regional Levels,* publication #495, Centre de recherche sur les transports, Université de Montréal, 1987.

Florian, M., Crainic, T.G. (Editors), *Strategic Planning of Freight Transportation in Brazil,* Final Report, Centre de recherche sur les transports, Université de Montréal, 1989.

Foley, J.D., Van Dam, A., Fundamentals of Interactive Computer Graphics, Addison Wesley, 1984.

Gallo, G., *Reoptimization Procedures in Shortest Path Problems,* publication #132, Centre de recherche sur les transports, Université de Montréal, 1981.

Gallo, G., Pallottino, S., *A New Algorithm to Find the Shortest Paths between all Pairs of Nodes,* Discrete Applied Mathematics 4, 1982.

Gallo, G., Pallottino, S., *Shortest Path Methods: A Unifying Approach,* Mathematical Programming Study 26, 1986.

Gallo, G., Pallottino, S., *Shortest Path Algorithms,* Anals of Operations Research, Fortran Codes for Network Optimization 13, 1988.

Garey, M.T., Johnson, D.S., Computers and Intractability: A Guide to the Theory of NP Completeness, W.H. Freeman, San Francisco, 1979.

General Motor Transportation System Center, *Interactive Graphic Transit Design System (IGTDS): User's Manual,* General Motors Technical Center, Warren, Miami, 1978.

Gilsinn, J., Witzgall, A., *A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees,* National Bureau of Standards, Technical Note N. 772, 1973.

Glover, F., Glover, R., Klingman, D., *Computational Study of an Improved Shortest Path Algorithm,* Networks 14, 1984.

Glover, F., Klingman, D., Phillips, N., *A New Polynomially Bounded Shortest Path Algorithm,* Operations Research 33(1), 1985(a).

Glover, F., Klingman, D.D., Phillips, N.V., Shneider, R.F., *New Polynomial Shortest Path Algorithms and their Computational Attributes,* Management Science 31(9), 1985(b).

Gondran, M., Minoux, M., Graphes et algorithmes, Éditions Eyrolles, Paris, 1985.

Guélat, J., Florian, M., Crainic, T.G., *A Multimode Multiproduct Network Assignment Model for Strategic Planning of Freight Flows,* publication #549, Centre de recherche sur les transports, Université de Montréal, 1987.

Hasselstrom, D., *Public Transportation Planning - A Mathematical Programming Approach",* Ph.D. thesis, Department of Business Administration, University of Gothenburg, Suède, 1981.

Horowitz, A.J., Pithavadian, A.R., *Generalized Computer Aided Design of Transportation Networks,* Transportation Quarterly *41(3), 397-409, 1987.*

Hung, M.S., Divoky, J.J., *A Computational Study of Efficient Shortest Path Algorithms,* Computer and Operations Research 15(6), 1988.

Johnson, D.B., *A Note on Dijkstra's Shortest Path Algorithm,* J.A.C.M. 20(3), 1973.

Johnson, E.L., *On Shortest Paths and Sorting,* Proceedings of the A.C.M. Annual Conference, Boston, 1972.

Kennington, J.L., Helgason, R.V., *Algorithms for Network Programming,* John Wiley & Sons, 1980.

Kernigham, B.W., Ritchie, D.M., *The State of C,* BYTE, McGraw-Hill publication 13(8), août 1988.

Kernigham, B.W., Ritchie, D.M., Le langage C, Masson, Paris, 1983.

Kershenbaum, A., *A Note on Finding Shortest Path Trees,* Networks 11, 1981.

Kirby, R.F., Potts, R.B., *The Minimum Route Problem for Networks with Turn Penalties and Prohibitions,* Transportation Research 3, 1969.

Kornhauser, A.L., Hess, J.L., *Interactive Graphics Sketch Planning Model for Urban Transportation,* Transportation Research Record 657, 1978.

Kornhauser, A.L., *Examples of Corporate Distribution and Transportation Planning Using the Princeton Transportation Network Model and Graphic Information System,* Proceedings of the Babson College Conference on Computer Software Uses in Transportation and Logistics Management, Babson College, Babson Park, Massachusetts, 1984.

Kornhauser, A.L. *Examples of Corporate Distribution and Transportation Planning Using the Princeton Transportation Network Model and Graphic Information System,* Civil Engineering Department, Princeton University, 1985(a).

Kornhauser, A.L., *Very Large Transportation Network Interactivem Computer Graphic Information System and Problem Solver,* TIMS/ORSA Joint National Meeting, Boston, 1985(b).

Latour, D., *Une étude des méthodes de calcul des k meilleurs chemins,* Mémoire de maîtrise, Département d'informatique et de recherche opérationnelles, Université de Montréal, juillet 1987.

Lawler, E.L., Combinatorial Optimization: Networks and Matroids, Holt Rinhart and Winston, New York, 1976.

Lebeuf, D., *Allocation des appareils pour un horaire aérien : une approche interactive-graphique,* publication #415, Centre de recherche sur les transports, Université de Montréal, 1985.

Lutin, J.M., Hazony, Y., Kornhauser, A.L., *Interactive Network Model for Analysis and Cartographic Display of Railroad Traffic Flow,* T.R.B. Transportation Research Record 729(1-7), 1979.

Mondou, J.-F., Crainic, T.G., *Conception d'un logiciel interactif-graphique pour la planification tactique du transport des marchandises,* Journées de l'optimisation, Montréal, 1988.

Mondou, J.-F., *Chemins les plus courts : nouvelles structures de données, nouveaux algorithmes,* Journées de l'optimisation, Montréal, 1989.

Mondou, J.-F., Crainic, T.G., Couture, S., *BGC - Bibliothèque de fonctions Graphiques en langage C,* publication #554, Centre de recherche sur les transports, Université de Montréal, 1987.

Mondou, J.-F., Crainic, T.G., Nguyen, S., *Shortest Paths Revisited: The Impact of the C Language Programming on Efficiency,* CORS/TIMS/ORSA Joint National Meeting, Vancouver, 8-10 mai 1989.

Pallottino, S., *Adaptation de l'algorithme de d'Esopo-Pape pour la détermination de tous les chemins les plus courts : améliorations et simplifications,* publication #136, Centre de recherche sur les transports, Université de Montréal, 1979.

Pallottino, S., Storchi, G., *The Computational Complexity of Two Tree-Indexing Methods,* Istituto Per Le Applicazioni del Calcolo "Mauro Picone", Serie III, No. 206, Rome, 1981.

Pallottino, S., Storchi, G., *On the Mean Complexity of Tree-Handling Techniques in Network Programming,* Istituto per le Applicazioni del Calcolo "Mauro Picone", Serie III, No. 175, Rome, 1984.

Pape, U., *Implementation and Efficiency of Moore's Algorithm for the Shortest Route Problem,* Mathematical Programming 7, 212-222, 1974.

Pape, U., *Algorithm 562: Shortest Path Lengths,* A.C.M., Transactions on Mathematical Software 6(3), 1980.

Pearson, R., *Aspects avancés du langage C, Interédition, Paris, 1987.*

Pollack, M., Wiebenson, W., *Solutions of the Shortest-Route Problem - A Review,* Operations Research 8, 1960.

Rapp, M.H., Mattenberger, P., Piguet, S., Robert-Grandpierre, A., *Interactive Graphic System for Transit Route Optimization,* Transportation Research Record 619, 1976.

Riepl, R.J., Gehner, C.D., *Interactive Graphics in Europe: Three Computer-Aided Transportation Planning tools,* Computer Graphics 2(5), 30-36, 1979.

Roland Digital Group, *Drafting Plotter GRX-400/300,* Operation Manual.

Roy, B., Algèbre moderne et théorie des graphes, deuxième tome, DUNOD, Paris, 1970.

Roy, J., *Un modèle de planification globale pour le transport routier des marchandises,* thèse de doctorat en administration, École des Hautes Etudes Commerciales, Montréal, publication #402, Centre de recherche sur les transports, Université de Montréal, 1984.

Roy, J., Delorme, L., *NETPLAN: A Network Optimization Model for Tactical Planning in the Less-than-Truckload Motor Carrier Industry,* INFOR 27(1), 1989.

Sauvageau, B., *Codification d'algorithme de recherche des chemins minimaux d'un réseau",* publication #264, département d'informatique et de recherche opérationnelle, Université de Montréal, janvier 1978.

Sedgewick, R., Vitter, J.S., *Shortest Paths in Eucledean Graphs,* 25th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1984.

Shier, D., Witzgall, C., *Properties of Labeling Methods for Determining Shortest Path Trees",* J. Res. National Bureau of Standards 86, 1981.

Standish, T.A., Data Structure and Techniques, Addison-Wesley, 1980.

Spiess, H., *Contribution à la théorie et aux outils de planification de réseaux de transport urbain,* thèse de doctorat, département d'informatique et de recherche opérationnelle, Université de Montréal, publication #382, Centre de recherche sur les transports, 1984.

Tarjan, R.E., *Data Structures and Network Algorithms,* Society for Industrial and Applied Mathematics, Philadelphia, 1983.

Wolcott, N.M., Hilsenrath, J., *A Contribution to Computer Typesetting Techniques: Tables of Coordinates for Hershey's Repertory of Accidental Typefonts and Graphic Symbols,* National Bureau of Standards, PB 251-845, 1976.

Yen, J.Y., *An Algorithm for Finding Shortest Routes from all Source Nodes to a Given Destination in General Network,* Quart. Appl. Math. 27, 1970.

REMERCIEMENTS

En premier lieu, j'exprime ma gratitude envers mon directeur de recherche, Teodor Gabriel Crainic, professeur agrégé au département des Sciences administratives de l'Université du Québec à Montréal, et chercheur au Centre de recherche sur les transports de l'Université de Montréal, ainsi qu'à Sang Nguyen, professeur titulaire au département d'Informatique et de Recherche opérationnelle de l'Université de Montréal, et chercheur au Centre de recherche sur les transports, qui a accepté de codiriger ce travail. Ils m'ont témoigné, tout au long de ces dernières années, leur intérêt, leurs commentaires judicieux, leur confiance et leur disponibilité.

Je tiens également à remercier :

- Lucie L'Heureux pour son travail de dactylographie;
- le professeur Teodor Gabriel Crainic, l'organisme F.C.A.R., de même que le Centre de recherche sur les transports pour leur appui financier;
- le projet Franco-Québécois 20020686FQ pour les deux stages de perfectionnement que j'ai pu réaliser à l'École Centrale de Paris.